基于 BootStrap3 的 JSP 项目实例教程

主　编　李明欣　林　琳
副主编　朱卫平　康　凤

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

基于 BootStrap3 的 JSP 项目实例教程

主　编　李明欣　林　琳
副主编　朱卫平　康　凤

北京航空航天大学出版社

内 容 简 介

《基于 BootStrap3 的 JSP 项目实例教程》是作者多年来教学实践经验的总结，汇集了作者在项目中遇到的各种问题及解决方案。

本书采用迭代的方式讲解，以实际项目中的增删改查为基础，采用不同的技术迭代实现，在这个过程中引入新技术：预编译、连接池、事务、存储过程、JavaBean，Servlet 技术、BootStrap3 前端技术。最终达到使用 JSP＋JavaBean＋Servlet＋BootStrap3 实现 MVC 的开发实际项目，并在最后一章提供了综合案例——博客系统。本书是一本注重实际操作的实例教程，在讲解的过程中，只需掌握基本的理论，然后通过实战案例就能轻松掌握。

《基于 BootStrap3 的 JSP 项目实例教程》随书带有源码、课件、教案、开发工具，以方便学习以及相关的学习视频，读者可以到作者的百度云下载，链接地址：http://pan.baidu.com/share/home?uk=3473194016。

《基于 BootStrap3 的 JSP 项目实例教程》内容全面，结构清晰，注重实战，非常适合 Java Web 开发人员学习使用，同时也可以作为软件公司的参考书。

图书在版编目(CIP)数据

基于 BootStrap 3 的 JSP 项目实例教程 / 李明欣，林琳主编. -- 北京 ：北京航空航天大学出版社，2015.8

ISBN 978-7-5124-1821-9

Ⅰ. ①基… Ⅱ. ①李… ②林… Ⅲ. ①JAVA 语言—程序设计—高等学校—教材 ②JAVA 语言—网页制作工具—高等学校—教材 Ⅳ. ①TP312②TP393.092

中国版本图书馆 CIP 数据核字(2015)第 151122 号

基于 BootStrap3 的 JSP 项目实例教程

主　编　李明欣　林　琳

副主编　朱卫平　康　凤

责任编辑　罗晓莉

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191)　http://www.buaapress.com.cn

发行部电话：(010)82317024　传真：(010)82328026

读者信箱：bhpress@263.net　邮购电话：(010)82316936

北京时代华都印刷有限公司印装　各地书店经销

*

开本：787×1 092　1/16　印张：15.5　字数：397 千字

2015 年 8 月第 1 版　2015 年 8 月第 1 次印刷　印数：3 000 册

ISBN 978-7-5124-1821-9　定价：32.00 元

前 言

本书以“突出技术，强化能力，综合应用”为指导思想，通过项目驱动，以多个实际应用案例为引导，介绍 JSP＋Servlet＋JavaBean 以及页面前端 BootStrap3 框架技术。书中的技术介绍、实际项目载体以及经典模块代码都来源于编者多年的教学与开发工作积累，具有实践性和操作性，从而帮助读者尽快掌握这些技术框架的使用方法。本书的特色如下：

1. 内容编排由浅入深，知识点新颖，紧跟时代潮流

本书章节按照由浅入深、循序渐进的顺序编排。内容选取上，精选 Java Web 技术中的流行技术，理论知识坚持够用原则，项目选择注重实用性和代表性。书中介绍的具体案例实现过程，也是培养读者实际参与项目开发的能力，适应社会，技术提高的过程。

2. 实例丰富，具有实操性

本书每章的知识点都配备了大量的应用实例，这些实例充分展现了相关知识点的实现细节。读者可以在学习相关知识点后，结合实践来更形象深入地了解并运用这些知识点。案例的实现，也是知识融会贯通的过程。

3. 提供了完整的项目实例，培养读者综合应用能力

本书以当前 Java Web 技术的主流开发技能需求出发，前台页面采用 BootStrap3 框架，结合使用 JSP＋JavaBean＋Servlet 技术，实现了完整的博客系统。通过对项目深入研究，读者可以比较全面地掌握基于 Java Web 应用程序的开发步骤和开发方法，并可将实例中所采用的技术迁移应用到自己的项目中。

4. 配有源代码与教学视频，辅助读者自学

为了方便读者掌握本书提供的实例，所有的源代码共享到百度云。另外还提供了高清教学视频，将书中的一些相关操作直观地展示给读者，以帮助读者达到更好的学习效果。

本书以项目驱动方式讲解如何使用 JSP 和页面前端框架 Bootstrap3 开发 Java Web 应用。第 1 章介绍配置和开发 JSP 应用程序；第 2 章讲解 JSP 的常用语法；第 3 章讲解 JSP 常用对象；第 4 章讲解 JDBC 数据库的访问技术；第 5 章讲解 JSP 中调用 JavaBean 的基本方式以及如何使用 JavaBean 进行抽象，简化 JSP 的开发；第 6 章讲解 Serlvet 常用的 API 以及通过 Servlet 访问数据库的方法，并最终实现 JSP＋JavaBean＋Servlet MVC 网站设计；第 7 章在前面的基础上进一步添加 Ajax 技术；第 8 章讲解流行的 Web 前端框架 BootStrap3；第 9 章提供了一个完整的博客系统，将前面的知识点全部整合。

本书由李明欣、林琳主编，朱卫平、康凤担任副主编，四川航天职业技术学院王立波担任主审，成都航空职业技术学院王津、蒋小惠，成都工业学院冯海波、南京工业职业技术学院查英华等人参与了部分章节的编写工作。

本书编写过程中参考了众多的技术开发门户网站，包括 www. csdn. net、www. iteye. com 和 www. gitnub. com 等，并汲取了多方人士的宝贵经验，在此向这些文献的作者和给予帮助的同仁们表示感谢。

由于编者水平有限，书中不妥之处恳请各位专家、老师及广大读者予以批评指正。

目　录

第1章 JSP开发入门

1.1 HTTP相关概念

HTTP(Hyper Text Transfer Protocol)是基于请求/响应模式的应用层协议,即客户端和服务器端不需要建立持久的连接,客户端提交请求,服务器端接收请求后返回响应,之后就断开连接。

客户端发出的请求头包含请求的方法、统一资源标识符(URI)、协议版本号、请求修饰符、客户端信息和内容类似 MIME 的消息结构。服务器以一个状态行作为响应,响应的内容包括消息协议的版本、成功或者错误编码、服务器信息、实体元信息以及可能的实体内容。

HTTP 还是一种无状态协议,体现在其对于事务处理无记忆能力。如果后续处理需要前面的信息,则它必须重传,这样可能导致每次连接传送的数据量增大。当然,在服务器不需要先前信息时,它的应答就较快。

HTTP 的请求/响应消息如果没有发送并传递成功,是不保存任何已发送出的信息的。比如,单击"提交"按钮,如果表单传递未成功,则浏览器将显示错误信息页,并返回空白表单。尽管没有发送成功,HTTP 也不保存表单信息。

基于 HTTP 协议的上述特点,通常,客户端每次需要更新信息都必须重新向服务器发起请求,客户端收到服务器返回的信息后再更新屏幕内容。

基于 HTTP 协议的客户端/服务器请求响应机制的信息交换过程主要包括四个步骤:

① 建立连接:客户端与服务器建立 TCP 连接。

② 发送请求:打开一个连接后,客户端把请求消息发送到服务器的相应端口上,完成请求动作提交。

③ 发送响应:服务器在处理完客户端请求之后,要向客户端回送响应消息。

④ 关闭连接:客户端和服务器双方都可以通过关闭套接字来结束 TCP/IP 对话。

如图 1-1 所示,HTTP 的工作机制就是请求和响应消息。简单说来就是一个用户输入一个站点地址,发送一个请求,之后,浏览器返回所请求的页面,这个页面可能是简单的 HTML 页面,也可能是动态编译后的页面。如果这个页面有错误或者不存在,则 Web 服务器将发送错误信息页面。

Web 服务器发送错误信息页是因为 HTTP 没有内置的处理机制,是无状态的,传输协议不记忆从一条请求消息到另一条请求消息的任何信息。这个特点可以保证 Web 的一致性。但是,用户常常需要记忆一些设置内容或者浏览过程,这就需要在 Web 页面或者 URL 中携带各种参数及其值。

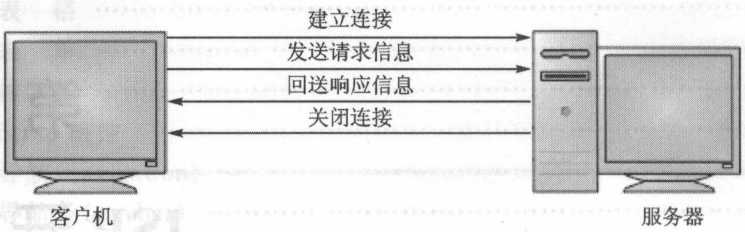


图 1-1　客户端/服务器信息交换过程

1.2　Servlet 的优势与问题

Servlet 是用 Java 编写的在服务器上运行的一个应用程序，负责处理客户端请求的信息并回送。Servlet 延续了 Java 在跨平台上的表现，完全可以在 Apache 等不同 Web 服务器上执行，无论底层的操作系统是 Windows、Solaris、Mac、Linux，还是其他的能支持 Java 的操作系统。

Servlet 处理请求的效率高于 CGI。Servlet 在加载执行之后，其对象实体通常会一直停留在 Server 的内存中，若有请求(request)发生，服务器再调用 Servlet 来服务，当收到相同服务的请求时，Servlet 会利用不同的线程来处理，而 CGI 程序必须启动许多进程 (process)来处理数据。

Servlet 借助 Java 的强大功能，不仅能简单处理客户端的请求，还可以实现大量的服务器端的管理维护功能，以及各种特殊的任务。比如，并发处理多个请求、转送请求、代理等。

Servlet 可以和 HTML 代码配合使用实现 Java 代码。这个特点既是 Servlet 的优点，也是 Servlet 技术面临的一个常见问题，即在 Servlet 中内嵌 HTML 代码。如下面一个典型的 Servlet 代码。

```
package action;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet {
    private static final longserialVersionUID  = 1L;
    public FirstServlet() {}
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Servle-
tException, IOException {
    this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle-
tException, IOException {
    PrintWriter out = response.getWriter();
```

```
    out.println("<html><head>");
    out.println("<title>First Servlet </title>");
    out.println("</head><body>");
    out.println("hello servlet");
    out.println("</body></html>");
    out.close();
    }
}
```

在 doPost 方法里，out. print()调用包含着许多 HTML 标记。这种方式不适合输出大量的 HTML 代码。对于要实现一个简单 Web 站点的个人开发者来说这种方式可行，但当几个拥有不同技能的人协同开发 Web 应用程序时，这就变得极为困难。如今，网站变得越来越复杂，随之而来的问题是，对 Web 界面的美观性和易用性要求变高，服务器端代码更加复杂等。人们期待有一种开发模型，可以让拥有不同技能的人在一起高效率地协同完成这些复杂的 Web 应用程序。JSP(Java Server Pages)所提供的正是这样一种开发模型，它使得擅长客户端技术的网页设计师与精通服务器端技术的程序员能够协调地工作。

尽管 Servlet 的功能非常强大，但它通常属于程序员专有的领域。实际运用中，实现 Servlet 方案需注意以下问题：

① 开发和维护应用程序都要求程序员具有深厚的 Java 编程基础，因为处理代码和 HTML 元素是交织在一起的。

② 改变应用程序的外观和风格，或者加入对某种新类型客户机的支持时，都需要更新并重新编译 Servlet 代码。

③ 很难利用网页开发工具的优势来设计应用程序界面。如果使用这些工具来开发网页布局的话，生成的 HTML 代码必须被手工嵌入到 Servlet 代码中，这个过程既耗时又容易出错，而且极度枯燥乏味。

将 Servlet 和 JSP 页面组合起来使用，可以实现应用程序的业务逻辑处理和界面显示相对独立。

1.3　JSP 介绍

1.3.1　JSP 简介

JSP 是由 Sun Microsystems(已被 Oracle 收购)公司倡导、许多公司参与共同建立的一种动态页面技术标准。JSP 页面由模板文本和嵌入其中的 Java 代码以及 JSP 元素所组成。JSP 页面中任何不是 JSP 元素的东西都可称为模板文本(template text)，模板文本一般直接传送给浏览器，它可以是任何文本：HTML、XML、甚至纯文本。由于 HTML 是目前最常用的网页语言，因此本书中绝大多数叙述和示例都使用了 HTML。但 JSP 并不依赖 HTML，它可以与任何一种标记语言共同使用。

对 Web 服务器中 JSP 的访问过程是：客户端发出 JSP 页面访问请求，服务器首先执行其中的程序段，然后将执行结果连同 JSP 文件中的模板文本动态合成后的页面返回给客户端。

所有程序操作都在服务器端执行，网络上传送给客户端的仅是得到的结果，而结果通常就是 HTML 页面，因此客户端只要有浏览器即使不支持 Java 也能浏览。JSP 页面一般很少进行数据处理，只是用来实现网页的静态化页面和提取数据，不会进行业务处理。

Java Servlet 是 JSP 的技术基础，它和 JSP 配合才能完成一些大型的 Web 应用程序开发。JSP 本身也继承了 Java 技术简单易用、完全面向对象、跨平台且安全可靠、主要面向 Internet 等大部分优点。JSP 技术将网页逻辑与网页设计的显示分离，支持可重用的基于组件的设计，使基于 Web 的应用程序的开发变得简单和高效。

自 JSP 推出后，众多大公司都相继推崇支持 JSP 技术的服务器，如 IBM、Redhat 公司等，因此 JSP 迅速发展成为商业应用的服务器端语言。实现和 1.2 节 Servlet 代码相同功能的 JSP 页面的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>Hello Servlet</title>
</head>
  <body>
    <% out.print("hello servlet"); %>
  </body>
</html>
```

其中需要编写的代码只有<% out.print("hello servlet"); %>，相对于 Servlet 来说，JSP 简化了代码的编写。

1.3.2 JSP 的处理过程

在处理一个 JSP 页面请求时，模板文件与由 JSP 元素产生的动态内容组合起来，组合的结果将作为应答发回给浏览器。整个处理过程通常可分为两个阶段：翻译时期(translation time)和请求时期(request time)。详细的处理过程分为 2 步。

① 用户发出 JSP 页面请求。如果此页面是第一次被请求，JSP 引擎会通过预处理把 JSP 源文件中的静态数据(HTML 文本)和动态数据(Java 脚本)全部处理成 Servlet 代码(相应的.java)，并将生成的 Servlet 代码编译成 Servlet 类文件(相应的.class)。对于 Tomcat 服务器而言，生成的类文件默认的路径是存放在<Tomcat>\work 目录中。

② 在不修改 JSP 页面前提下，如果此页面不是第一次被请求，任何后续的请求都会直接进入请求处理阶段，即 JSP 引擎会直接运行 Servlet 类文件，并将处理结果返回给用户。而当 JSP 页面被修改后，后续请求还需再次通过翻译阶段，才能进入请求处理阶段。

如上所述，执行 JSP 网页的过程通常可分为两个时期：请求时期(Request Time)和翻译时期(Translation Time)。在翻译时期，JSP 网页被翻译成 Servlet 代码，然后被编译成类文件。

当第一次收到页面请求，JSP 容器会自动翻译这个页面，这需要一些时间完成，所以用户第一次发出 JSP 页面请求后，会发现有延迟现象。用户也可采用 JSP 的页面进行预编译避免延迟的问题。这两个时期如图 1-2 所示。

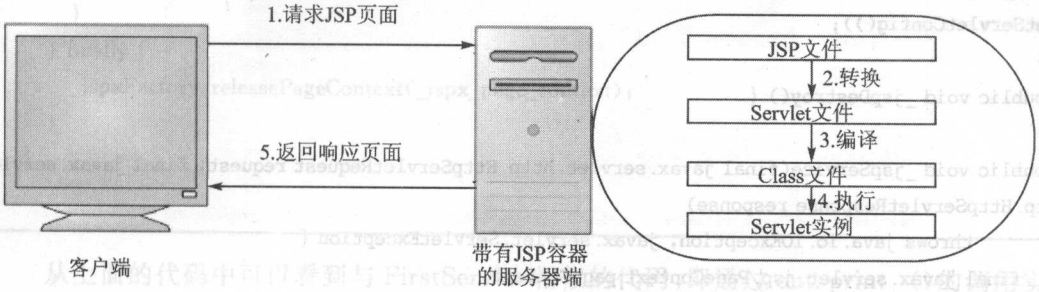


图 1-2　请求时期和翻译时期

JSP 容器通常以 Servlet 的形式来实现，经过配置，可以处理所有对 JSP 页面的请求。事实上，Servlet 容器和 JSP 容器这两种容器通常都集成于一个名为 Web Container 的软件包中。当然，JSP 页面成为编写 Servlet 的另外一种方式，使得具备一定 Java 基础的从业者，也能很快上手。而且，除了翻译阶段，JSP 页面的处理过程与常见的 Servlet 完全相同：即载入一次后多次重复执行，直到服务器关闭为止。作为一种自动生成的 Servlet，JSP 页面继承了 Servlet 的所有优点：跨平台和厂商独立性、高集成性、高效性、可缩放性、健壮性和安全性等。

下面的代码是通过配置 Tomcat 查看 hello.jsp 生成的 Servlet 类。

```
 *
 * Generated by the Jasper component of Apache Tomcat
 * Version: Apache Tomcat/7.0.30
 * Generated at: 2013-12-23 07:58:32 UTC
 * Note: The last modified time of this file was set to
 *       the last modified time of the source file after
 *       generation to assist with modification tracking.
 */
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
  private static final javax.servlet.jsp.JspFactory _jspxFactory =
          javax.servlet.jsp.JspFactory.getDefaultFactory();
  private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;
  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.tomcat.InstanceManager _jsp_instancemanager;
  public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
    return _jspx_dependants;
  }
```

```
  public void _jspInit() {
    _el_expressionfactory = _jspxFactory.getJspApplicationContext(getServletConfig().getServlet-
Context()).getExpressionFactory();
    _jsp_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager
(getServletConfig());
  }
  public void _jspDestroy() {
  }
  public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.
http.HttpServletResponse response)
        throws java.io.IOException, javax.servlet.ServletException {
    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;
    try {
      response.setContentType("text/html; charset=UTF-8");
      pageContext = _jspxFactory.getPageContext(this, request, response,
          null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;
      out.write("\r\n");
      out.write("<! DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" \"http://www.
      w3.org/TR/html4/loose.dtd\">\r\n");
      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\r\n");
      out.write("<title>Hello Servlet</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");
      out.print("hello servlet");
      out.write("\r\n");
      out.write("</body>\r\n");
      out.write("</html>");
    } catch (java.lang.Throwable t) {
      if (! (t instanceof javax.servlet.jsp.SkipPageException)){
        out = _jspx_out;
```

```
            if (out ! = null && out. getBufferSize() ! = 0)
              try { out. clearBuffer(); } catch (java. io. IOException e) {}
            if (_jspx_page_context ! = null) _jspx_page_context. handlePageException(t);
            else throw new ServletException(t);
          }
        } finally {
          _jspxFactory. releasePageContext(_jspx_page_context);
        }
      }
    }
```

从上面的代码中可以看到与 FirstServlet 相似的代码，即通过 out. print ()的调用实现打印的 HTML 标记。上述语句代码中字体使用加粗斜体的就是使用<% out. print("hello servlet"); %>转成为 Servlet 后得到的代码。

注：要查看 JSP 编译后生成的类，需要对 tomcat 的 server. xml 进行配置，配置方法如图 1-3 所示，tomcat 与 eclipse 的开发配置会在第 2 章进行讲解。

```
        <!-- Access log processes all example.
             Documentation at: /docs/config/valve.html
             Note: The pattern used is equivalent to using pattern="common" -->
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory

      <Context docBase="Hello1" path="/Hello1" reloadable="true"
      source="org.eclipse.jst.jee.server:Hello1" workDir="d:/java/temp"/>
      </Host>
    </Engine>
  </Service>
</Server>
```

图 1-3　配置 server. xml

配置的代码如下。

```
<Context docBase = "Hello1" path = "/Hello1" reloadable = "true" source = "org. eclipse. jst. jee. server:Hello1"
workDir = "d:/java/temp"/></Host>
```

关键的属性是 workDir="d:/java/temp"，将会在 d:/java/temp 目录下生成如图 1-4 所示的类和 class 文件。

综上所述，JSP 具有如下优点：

① 跨平台性。作为 Java 平台的一部分，JSP 具有一次编写、在各平台上运行的特点。

② 系统的多平台支持。JSP 基本上可以在所有平台上开发、系统部署和扩展。

③ 强大的可伸缩性。从只有一个小的 Jar 文件就可以运行 Servlet/JSP，到由多台服务器进行集群和负载均衡，再到多台 Application 进行事务处理和消息处理，JSP 在一台服务器到无数台服务器上的能力展示，充分显示出了 Java 强大的生命力。

④ 开发效率高。JSP 应用充分支持 JavaBean 组件或 EJB(Enterprise JavaBean)组件，直接利用其经测试和信任的已有组件，根据需求再整合一些附加功能后，就可快速满足开发要求，避免了很多重复开发工作，有效提高应用程序的开发效率。

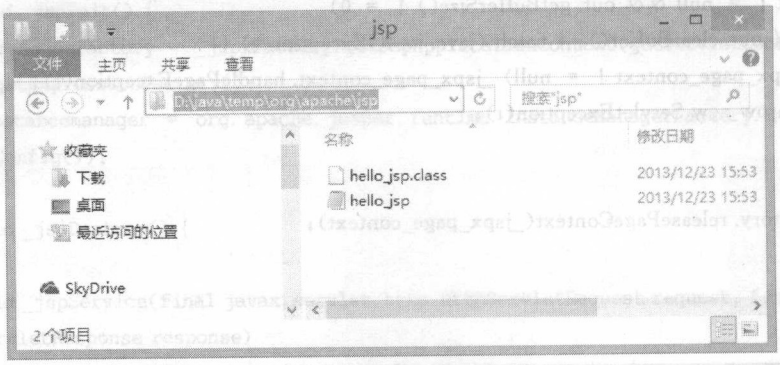


图 1-4　JSP 编译后生成的类

当然 JSP 也存在下面的一些弱势：

① 产品复杂性高。为了达到系统跨平台和高伸缩性，需要全面了解 Java 系统中开发的如 JRE、JEE、JavaBeans 等产品，只有会正确地综合运用和搭配他们，才能产生强大的效能。

② 硬件要求高。JSP 技术高效率的同时也增加了对机器硬件的要求：一方面，.class 类文件需常驻内存；另一方面，一系列的.java 文件和.class 文件及相应的版本文件还要占硬盘空间。这使得对机器硬件性能配置要求会比较高。

在了解了 JSP 技术原理之后，下面将从搭建 JSP 开发环境开始实践 JSP。内容包括 JDK 的安装、Java 环境变量的配置、Eclipse 和 Tomcat 的安装与配置。

1.4　Java 环境变量的设置

要成功运行编写的 JSP 程序，首先需要安装 JDK(Java Developmer kit)，并配置运行 Java 的环境变量，然后再安装 Eclipse IDE 集成开发工具，并在 Eclipse 中配置 JSP 的运行 Web 服务器 Tomcat。首先讲解 JDK 的环境以及 Java 运行环境变量的设置。

先根据操作系统的版本到 http://java.oracle.com 网站上下载最新版的 Java Development Kit，本教材中采用的是 64 位 Windows 8 操作系统，下载完成之后进行安装，安装的步骤如下。

① 双击运行下载的 jdk-7u45-Windows-x64，出现 JDK 安装向导，如图 1-5 所示。

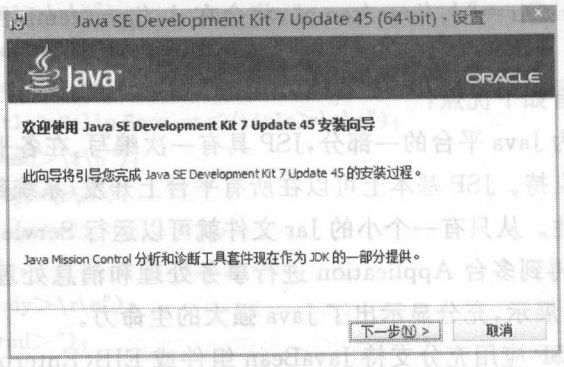


图 1-5　JDK 的安装步骤一

② 单击“下一步”按钮，选择“源代码”，将此功能安装在本地磁盘默认驱动器上，如图 1-6 所示。单击“更改”按钮可以设置安装路径。

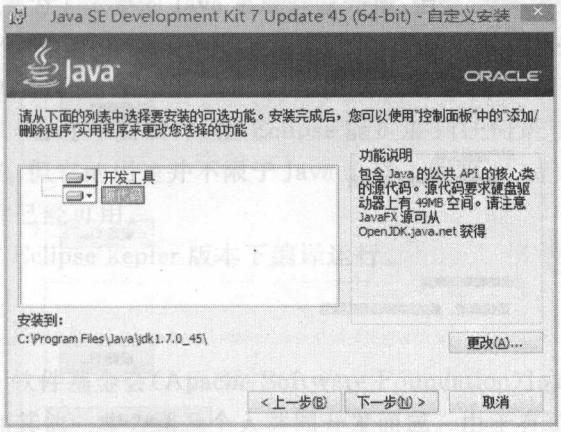


图 1-6　JDK 的安装步骤二

③ 单击“下一步”按钮开始安装 JDK，成功安装将显示如图 1-7 所示界面。单击“关闭”按钮完成 JDK 的安装。

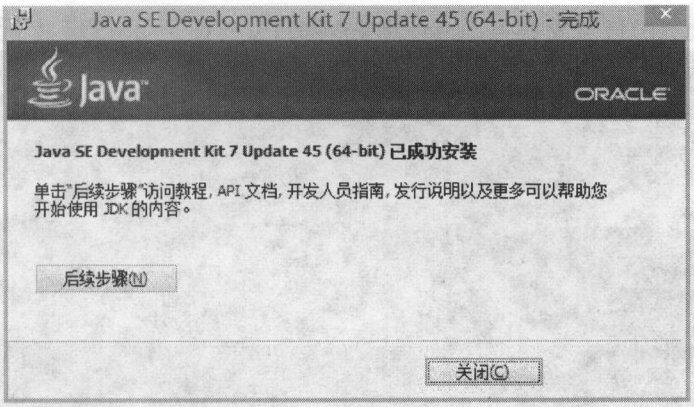


图 1-7　完成 JDK 安装

④ JDK 安装完成后开始配置 Java 程序设计的环境变量。在 Windows 8 中，打开文件资源管理器，右击“这台电脑”选择“属性”，或者通过控制面板/所有控制面板选项/系统/高级系统设置，出现“系统属性”对话框，如图 1-8 所示。

选择“高级”选项卡，单击右下角的“环境变量(N)”按钮，在系统变量中增加下面的选项：

- JAVA_HOME：C:\Program Files\Java\jdk1.7.0_45；
- CLASSPATH：.；%JAVA_HOME%\lib\dt.jar；%JAVA_HOME%\lib\tools.jar；

注意：在原有 CLASSPATH 后面增加，不可删除原有 CLASSPATH 内容。

- Path：；%JAVA_HOME%\bin；

注意：在原有 Path 后面增加，不可删除原有 Path 内容。

⑤ 验证环境变量是否设置成功：打开命令提示符，输入 javac，如果出现图 1-9 所示内容则表明 java 环境变量设置成功。

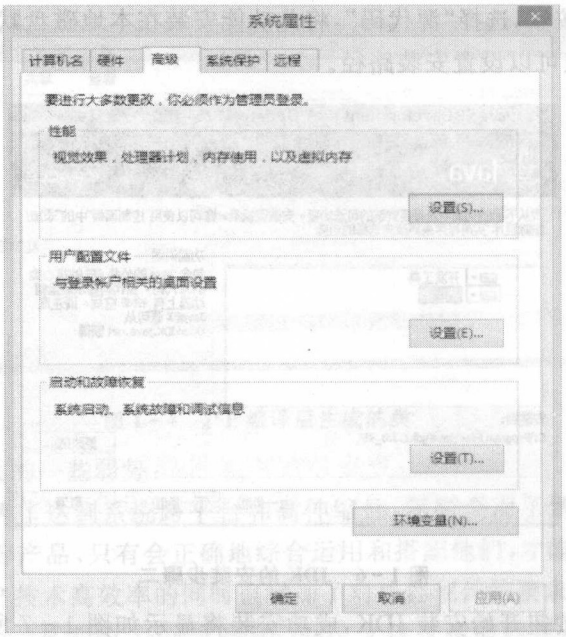


图 1-8　系统属性

图 1-9　验证环境变量设置是否成功

1.5　Eclipse 和 Tomcat 集成开发环境配置

1.5.1　Eclipse 简介

Eclipse 是由 IBM 公司牵头，多家公司和组织参与开发的一个开放源代码的、基于 Java 的可扩展开发平台，目前由 Eclipse 基金会管理。其本身只是一个框架和一组服务，用于通过插件或

组件构建开发环境。众多插件的支持使得 Eclipse 拥有其他集成开发环境(IDE)软件很难具有的灵活性,这个优势为它赢得了众多的支持者。许多软件开发商以 Eclipse 为框架开发自己的 IDE。

如今,Eclipse 已经成为了主要的 Java 集成开发环境,但它的应用目标不仅仅是提供一个专门开发 Java 程序的环境。Eclipse 除了包含有平台、开发工具箱,还包括有插件开发环境(Plug-in Development Environment,PDE)。也就是说对于高级软件开发人员来说,根据Eclipse的体系结构,可以通过开发插件,使 Eclipse 能扩展到任何语言的开发。尽管 Eclipse 是使用 Java 语言开发的,但它的用途并不限于 Java 语言。例如,支持诸如 C/C++、COBOL、PHP 等编程语言的插件已经可用。

本教材所有案例在 Eclipse kepler 版本下编译运行。

1.5.2 Tomcat 简介

Tomcat 是 Apache 软件基金会(Apache Software Foundation)Jakarta 项目中的一个核心项目,由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持,最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现。Tomcat 技术先进、性能稳定,并且免费开源,因此深受广大 Java 爱好者和程序员的喜欢,成为目前比较流行的 Web 应用服务器。

Tomcat 是一个小型的轻量级应用服务器,在中小型系统和并发访问用户不多的情况下,是开发和调试 JSP 程序的首选。可以简单这样理解,就是当在一台机器上配置好 Apache 服务器,可利用它响应对 HTML 页面的访问请求。Tomcat 是独立运行的 Apache 服务器的扩展,所以当运行 Tomcat 时,它实际上是作为一个与 Apache 独立的进程单独运行的。具体说来,Tomcat 就是一个支持 Servlet 和 JSP 容器运行的容器。对于 Web 服务器来说,Apache 仅支持静态网页,Tomcat 则不仅能为静态网页提供支持,同时还能提供动态网页服务,并为提高这种良好的支持而不断地扩充着。

Tomcat 的深受欢迎还源于它运行时占用的系统资源小,支持负载平衡与邮件服务等开发应用系统常用的功能,使得其扩展性好、安全性高;另外还具备更新快的特点,它一直在不断地改进和完善。任何一个有志于推动 Tomcat 发展的程序员都可加入更改或添加其新功能的行列。

截止 2015 年 7 月 31 日 Tomcat 最新版本为 8.0.24。本教材开发案例采用的 Tomcat 的版本号:7.0.42 免安装版。

1.5.3 开发调试环境的搭建

开发调试环境的搭建步骤为:

① 下载 Eclipse。使用浏览器登录 www.eclipse.org,单击 Download Eclipse,根据操作系统下载 64 位的 Eclipse IDE for Java EEDevelopers。

② 下载 Tomcat。登录 http://tomcat.apache.org/,下载 apache-tomcat-7.0.42 免安装版。

③ 解压 Eclipse 和 Tomcat。将下载好的 Eclipse 和 Tomcat 解压到自己的开发空间中,如 D:\JSP,解压 Eclipse 和 Tomcat 到该文件夹下。

④ 进入 eclipse 目录,双击 Eclipse 启动,启动的界面如图 1-10 所示。

⑤ 启动完成之后出现如图 1-11 所示界面,关闭 welcome 页面之后进入到图 1-12 的界面。

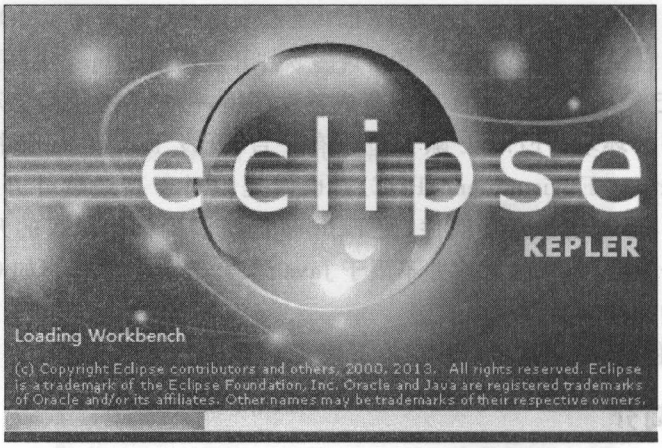


图 1-10　eclipse KEPLER 启动界面-1

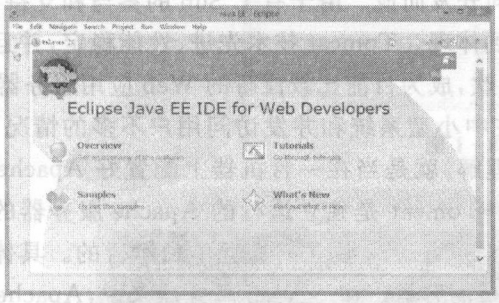


图 1-11　Eclipse 启动界面-2

图 1-12　Eclipse 启动界面-3

⑥ 选择菜单 Window/Prefercences，出现如图 1-13 所示的页面编码设置。选择 Prefercences 下的 Web 进入 JSP Files，将 Encoding 改为 UTF-8。

⑦ 设置服务器运行环境为 Tomcat。选择 Server/Runtime Environments，单击 Add，添加 7.0 的 Apache，如图 1-14 所示。然后选择步骤 3 解压的 apache-tomcat-7.0.42 的位置，如图 1-15 所示。

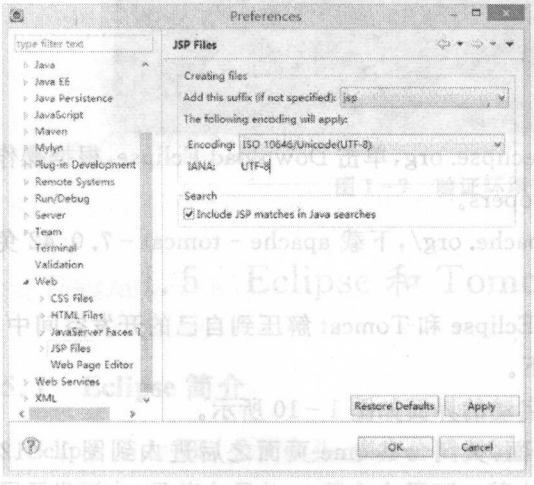


图 1-13　JSP 页面的编码设置

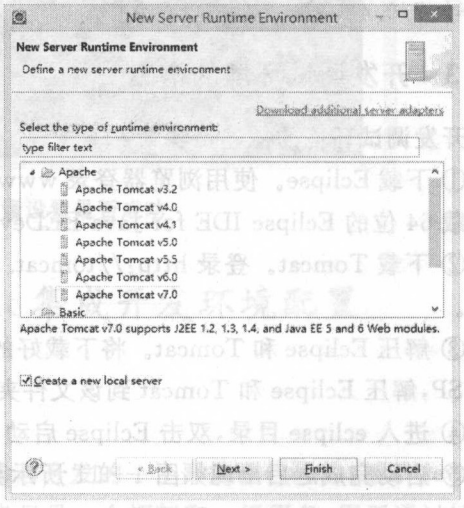


图 1-14　在 Eclipse 中添加 Tomcat 运行环境

⑧ 设置 Java Runtime Environment 运行环境。单击如图 1－15 所示的 Installed JRES 按钮，单击 Add，选择 Standard VM，并将 JRE home 改为安装 JDK 的位置，如图 1－16 所示。

⑨ 单击 Finish，JSP 的开发环境就配置完成。下面开始第一个 JSP 页面的编写。

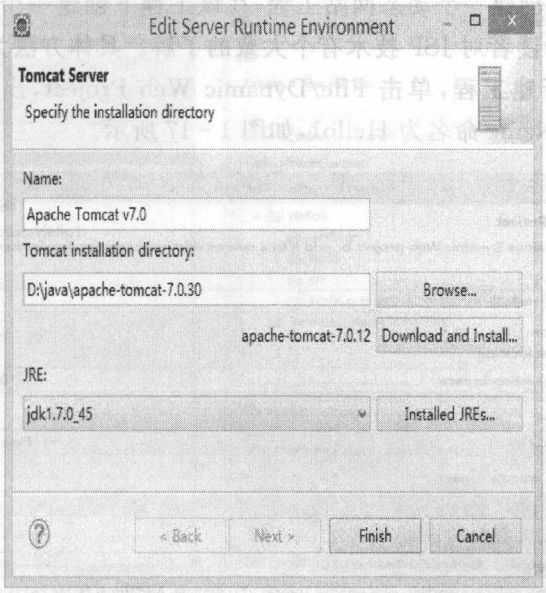


图 1－15　选择 Tomcat 的解压路径

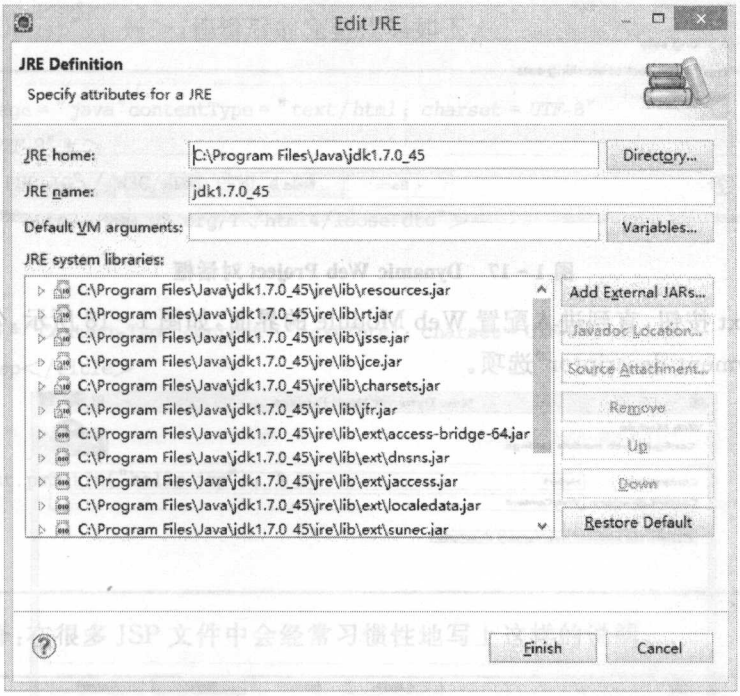


图 1－16　JRE 的配置

1.6 JSP 案例——Hello

下面将在 Eclipse 中创建一个动态网站工程，在该工程下创建一个简单 JSP 页面，用于输出“hello jsp”。从而帮助读者对 JSP 技术有个大致的了解。具体方法为：

① 创建 Project。新建工程，单击 File/Dynamic Web Project，出现 New Dynamic Web Project 对话框，Project Name 命名为 Hello1，如图 1-17 所示。

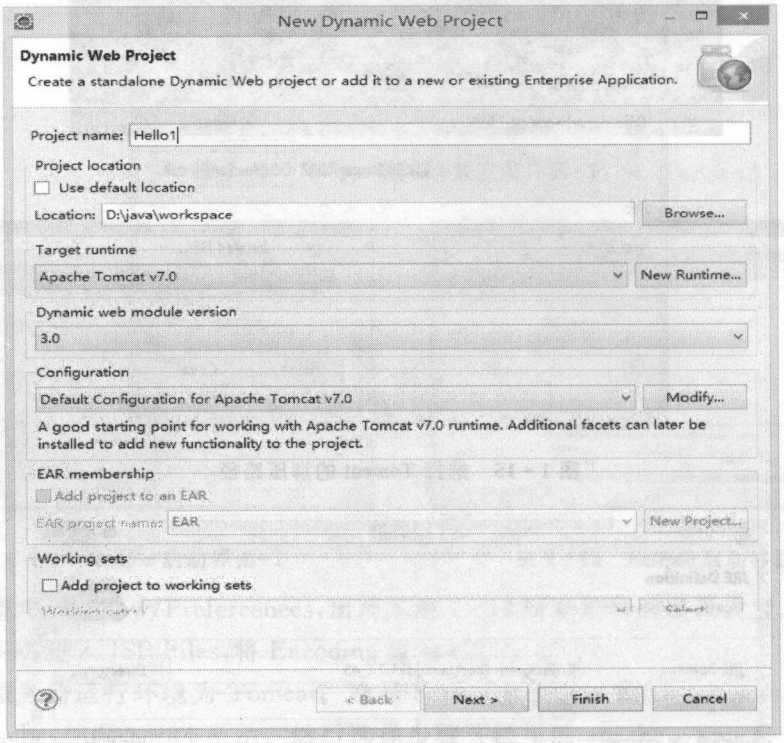


图 1-17 Dynamic Web Project 对话框

② 单击 Next 按钮，直到进入配置 Web Module 的界面，如图 1-18 所示，勾选“Genergate web. xml deployment descriptor”选项。

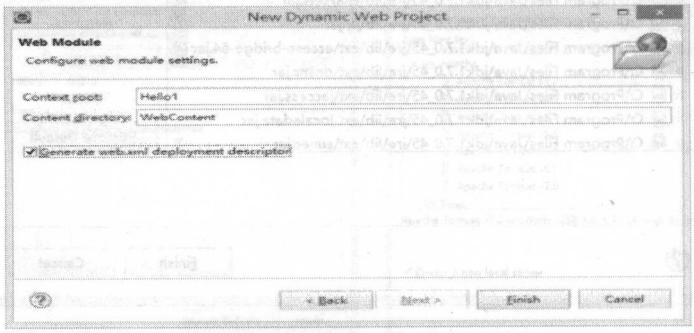


图 1-18 生成 web. xml

③ 单击 Finish，工程创建结束。在工程浏览器可以浏览工程树形结构，如图 1 - 19 所示。

④ 创建 JSP 页面。单击 WebContent，选择 New/JSP File，出现 New JSP File 向导，将第一个页面命名为 hello.jsp，如图 1 - 20 所示。单击 Finish 按钮，生成 JSP 文件。

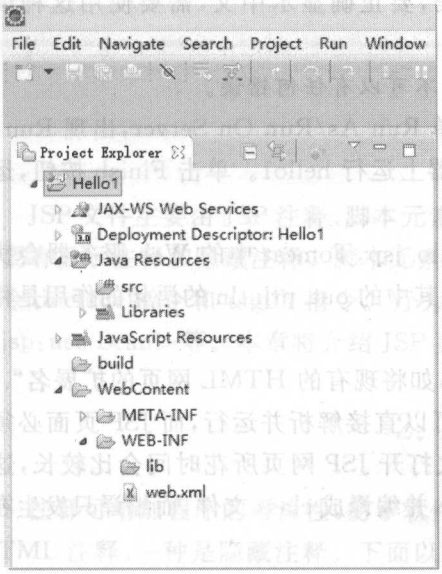


图 1 - 19　工程结构图

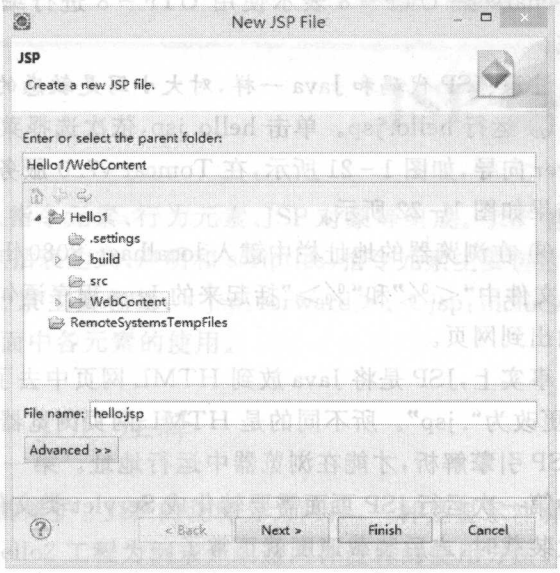


图 1 - 20　创建第一个 JSP 页面

⑤ 编辑 hello.jsp 页面，在＜title＞标签中录入“hello jsp”，在＜body＞标签中录入＜% out.println("hello jsp"); %＞，编辑后的完整代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>hello jsp</title>
</head>
    <body>
        <% out.println("hello jsp"); %>
    </body>
</html>
```

⑥ 代码解释：在很多 JSP 文件中会经常习惯性地写上这样的说明。

```
<%@page language = "java" contentType = "text/html; charset = UTF-8"
 pageEncoding = "UTF-8" %>
```

其中，language 属性声明了脚本语言的种类，该页面用了“java”。在目前的规范中，language 的属性默认为“java”，如果设置，也只能是“java”。

contentType 和字符编码相关。

charset=UTF-8 表示使用 UTF-8 进行编码，要正确显示中文，需要使用这种编码方式。

注意：JSP 代码和 Java 一样，对大小写是敏感的，不可以有任何错误。

⑦ 运行 hello.jsp。单击 hello.jsp，依次选择菜单 Run As/Run On Server，出现 Run On Server 向导，如图 1-21 所示，在 Tomcat v7.0 服务器上运行 hello1。单击 Finish 按钮，运行的结果如图 1-22 所示。

⑧ 在浏览器的地址栏中键入 localhast:8080/hello.jsp，Tomcat 中的 Web 服务器会执行 JSP 文件中“<%”和“%>”括起来的 Java 程序语句，其中的 out.println 的语句的作用是将文字输出到网页。

事实上，JSP 是将 Java 放到 HTML 网页中去了，如将现有的 HTML 网页的扩展名“.html”更改为“.jsp”。所不同的是 HTML 网页浏览器可以直接解析并运行，而 JSP 页面必须使用 JSP 引擎解析，才能在浏览器中运行地址。第一次打开 JSP 网页所花时间会比较长，这是因为第一次运行 JSP 页面需要转化成 Servlet 类文件，并编译成 class 文件，而编译只发生在第一次装载时，之后装载速度就正常了。

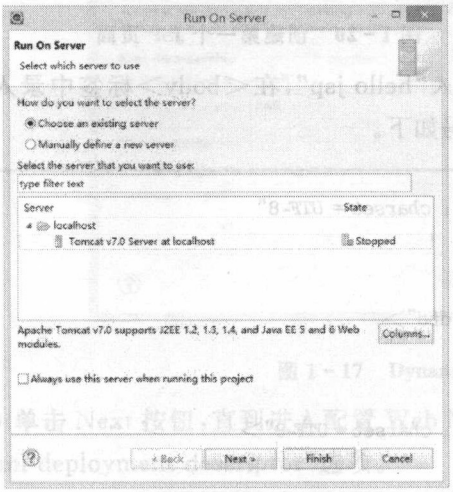


图 1-21　Run On Server 向导

图 1-22　hello.jsp 的运行结果

第2章 JSP语法

JSP文件主要由JSP注释、脚本元素、指令元素、行为元素、JSP对象等组成。JSP的注释主要有显示注释和隐藏注释。脚本元素包括表达式、声明和scriptles，指令元素主要包括page指令、include指令和taglib指令。行为元素主要包括：<jsp:forward>、<jsp:include>和<jsp:use Bean>等。本章将介绍JSP页面中各元素的使用。

2.1 JSP注释

注释可增加程序的可读性，易于软件的维护。JSP的注释分为两种，一种是显示注释又称HTML注释，一种是隐藏注释。下面以hello2工程为例来介绍这两种注释的使用方法。

① 创建工程。在Eclipse中创建工程名为hello2的Dynamic Web Project，在该工程下创建comment.jsp页面。

② 编辑comment.jsp页面，代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>注释</title>
</head>
<body>
<!-- 这行注释在浏览器查看源代码时能看见 -->          ---- HTML注释(输出注释)

<%--这行注释在浏览器查看源代码时不能看见--%>      ---- JSP页注释(隐藏注释)
</body>
</html>
```

代码解释。HTML注释(输出注释)：<!--注释-->生成的注释能够在客户端显示。标记内的所有JSP脚本元素、指令和行为正常执行，即编译器会扫描注释内的代码。

如：<!--这个页面加载于<%=(new java.util.Date()).toString()%>-->

在客户端的 HTML 源代码中显示为：

<！--这个页面加载于 SatMay0819:51:56CST2013-->

可以在注释中使用任何有效的 JSP 表达式。表达式是动态的，当用户第一次调用该页面或该页面后来被重新调用时，该表达式将被重新赋值。JSP 引擎对 HTML 注释中的表达式执行完后，其执行的结果将代替 JSP 语句。然后该结果和 HTML 注释中的其他内容一起输出到客户端。在客户端的浏览器中，浏览者可通过查看源文件的方法看到该注释。

JSP 页注释(隐藏注释)：<%--注释--%>用隐藏注释标记的字符会在 JSP 编译时被忽略掉，标记内的所有 JSP 脚本元素、指令和动作都将失效。即 JSP 编译器不会对隐藏注释符之间的任何语句进行编译，其中的任何代码都不会显示在客户端浏览器的任何位置。

JSP 引擎对 JSP 的这类注释(<%--注释--%>)不作任何处理，既不发送到客户端，也不在客户端的 JSP 页面中显示，在客户端查看源文件时也看不到。因此，当只想在 JSP 页面源程序中写文档说明时，此类 JSP 注释是很有用的。

运行的结果。运行的结果如图 2-1 所示。

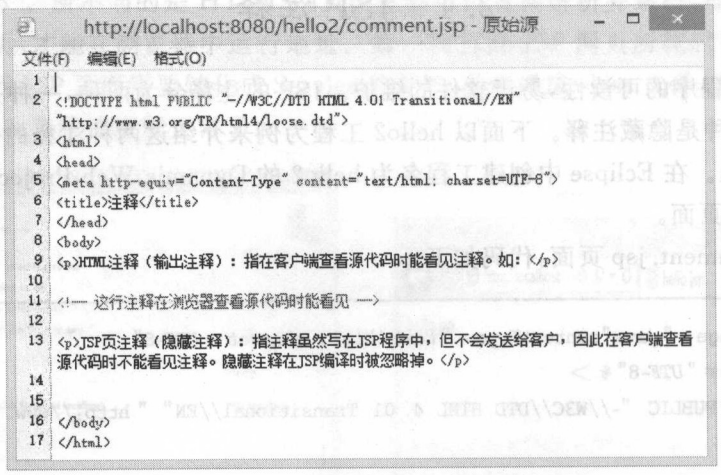


图 2-1　显示注释和隐藏注释的浏览器查看

2.2　脚本元素

JSP 页面中的脚本元素主要包括：声明(Declaration)、表达式(Expression)和小脚本(Scriptlets)，如表 2-1 所列。

表 2-1　JSP 的脚本元素

元　素	描　述
<%= %>	表达式，用于嵌入 Java 表达式，这些表达式的结果将加入应答中。也可以用作运行时行为的属性值
<%! %>	声明，用于在 JSP 页面的实现类中声明实例变量和方法
<% %>	小脚本，用于嵌入脚本代码

表中所列的脚本元素允许用户将小段的代码(一般情况下是 Java 代码)添加到 JSP 页面中(如可以加入一个 if 语句),以根据具体情况的不同而产生不同的 HTML 代码。同行为元素一样,它们也是在页面被请求时执行的。在使用脚本元素时需要格外小心,如果在 JSP 页面中加入了过多的代码,最后会遇到与将 HTML 嵌入 Servlet 时一样的维护问题。

2.2.1　scriptlet

本案例实现 1～100 累加求和的功能,以此展示 Scriptlets 脚本在 JSP 页面中的使用方法。

① 在 hello2 工程中创建 var.jsp 页面,编辑代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>1+2+3+...+100</title>
</head>
<body>
<%
    int sum = 0;//第 1 行代码
    for (int i = 0; i <= 100; i++)
        sum += i;
    out.print(sum);//第 4 行代码
%>
</body>
</html>
```

代码解释:Scriptlets 位于“<%”和“%>”之间,首先第 1 行代码定义了一个整形变量 sum,初始值为 0,其次采用 for 循环,不断地累加 sum,最终实现 sum=0+1+2+…+100。第 4 行代码在页面上打印出 sum 的值。

```
<%
int sum = 0;
for (int i = 0; i <= 100; i++)
sum += i;
out.print(sum);
%>
```

② 运行效果如图 2-2 所示。

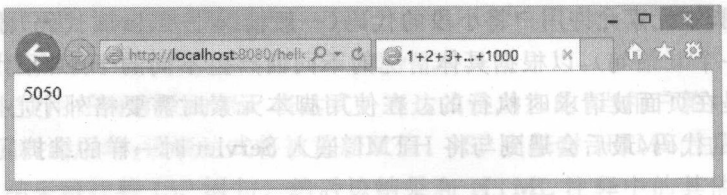


图 2-2　变量的定义与使用结果

2.2.2　表达式

本案例主要讲解表达式的使用方法，案例实现的主要功能是计算 $a+b=c$，并将 c 的值显示到页面上。

在工程 hello2 中创建 expression.jsp 页面，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>JSP 中变量的定义与使用</title>
</head>
<body>
    <%
        int a,b,c;//第 1 行代码
        a=10;//第 2 行代码
        b=15;//第 3 行代码
        c=a+b;//第 4 行代码
    %>
a+b=<%=c%>
</body>
</html>
```

代码解释：第 1 行代码了 3 个整形变量 a,b,c，第 2、3 代码将 a 的值赋为 10，b 赋值为 15，第 4 行代码将 a+b 和的值赋给 c。表达式是位于“<%=”和“%>”之间的代码，在本案例中为<%=c%>这段代码。

运行结果如图 2-3 所示。

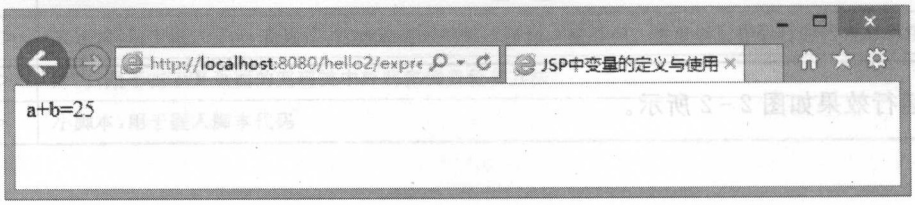


图 2-3　表达式的使用

2.2.3　声　明

JSP 声明用于在 JSP 程序中合法地声明变量、实例、方法和类。本案例主要介绍在声明下定义变量和在脚本中定义变量的区别。

在工程 hello2 中创建 count.jsp 页面，编辑的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>JSP</title>
</head>
<body>
<%!
    int i = 0;  //声明在<%! >内的变量
    %>
    <%
    int j= 0; //声明在<% >内的变量
    %>
    i的值:<% =  + +i %>
    j的值:<% =  + +j %>
</body>
</html>
```

当页面运行成功，多次刷新页面之后，大家会看见与图 2-4 相似的结果。为什么会出现这样的运行结果呢？原因在于声明在<%! %>内的变量和方法是一个类内的变量即成员变量(全局变量)，声明在<%%>内的变量是一个方法的变量(局部变量)。成员变量会被多个对象共享，而局部变量是单独对象访问。所以 i 会自动增加，j 不会递增。

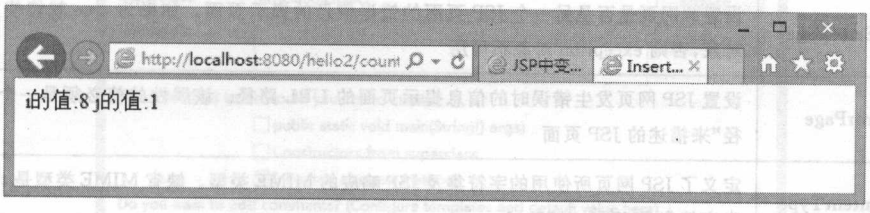


图 2-4　变量的两种定义方式

注：感兴趣的读者可以通过 JSP 的处理过程这节来验证，即通过生成 Java 类来验证上面所描述的内容。

2.3 指令元素

JSP 的指令元素主要包括:page 指令、include 指令以及 taglib 指令。它们都是 JSP 中的编译指令。编译指令就是告诉 JSP 的引擎,如何处理其他的 JSP 网页。JSP 编译指令的语法格式如下:

```
<%@ 指令名属性 = "属性值" %>
```

下面分别介绍 JSP 中的三种编译指令:Page 指令、include 指令以及 taglib 指令。

2.3.1 page 指令

功能:定义整个 JSP 页面的属性及其属性值。

语法格式如下:

```
<%@  page  属性 1 = "值"  属性 2 = "值" ... %>
```

所包含属性如表 2-2 所列。

表 2-2 page 指令的常用属性

序 号	属性值	属性值含义
1	language	定义 JSP 网页所使用的脚本语言的种类,其默认值是 java
2	import	指定 JSP 网页中需要导入的 java 包列表
3	buffer	设置此网页输出时所使用缓冲区的大小。buffer 的值可以为"none",也可以是一个数值。其默认值是 8Kb
4	autoFlush	指定当缓冲区满时是否自动输出缓冲区的数据(其值为布尔类型)。如果为 true,输出正常,否则当缓冲区满时将抛出异常。其默认值是 true。注意:如果把 buffer 的值设置为 none,那么把 autoFlush 的值设置为 false 就是非法的
5	info	指明网页的说明信息,可使用 Servlet 类的 getServletInfo 方法获取此信息
6	isErrorPage	设置此网页是否是另一个 JSP 页面的错误信息的提示页面。如果为 true 就能使用 exception 对象,否则 exception 对象不可用
7	errorPage	设置 JSP 网页发生错误时的信息提示页面的 URL 路径。该属性的值必须是一个用"URL 路径"来描述的 JSP 页面
8	contentType	定义了 JSP 网页所使用的字符集及 JSP 响应的 MIME 类型。缺省 MIME 类型是 text/html,缺省字符集是 ISO-8859-1

注意:page 指令作用于整个 JSP 页面以及由 include 指令和<jsp:include>包含进来的静态文件中,但不能用于动态包含文件。可以在一个页面上使用多个 page 指令,但是其中的属性只能使用一次(import 属性例外)。page 指令可以放在 JSP 文件的任何地方,但为了增强程序的可读性,好的编程习惯一般把它放在文件的顶部。

下面的案例实现通过 page 指令下的 import 属性导入 JavaBean。

① 在 Eclipse 中创建工程名为 hello3 的 Dynamic Web Project，展开 Java Resources，选中 src，右击，在弹出的菜单中选择 New，接着在二级菜单中单击 Class，如图 2-5 所示。

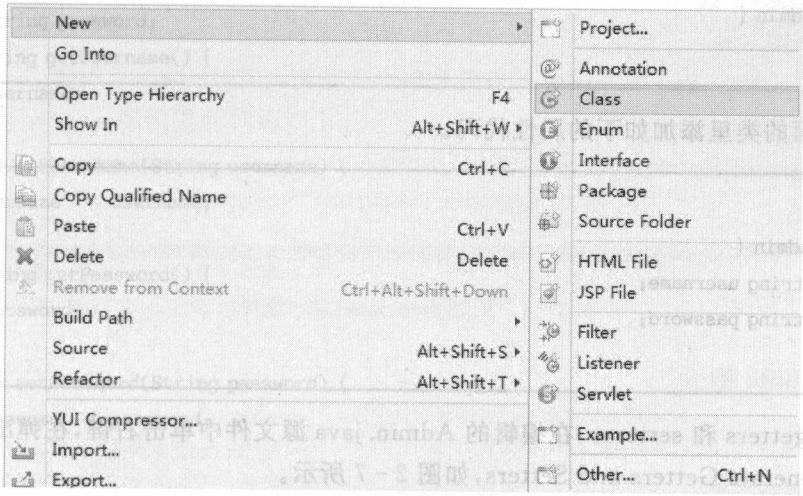


图 2-5　创建类菜单

② 在出现的新建类对话框中输入如图 2-6 所示的内容，Package 的值为 bean，Name 的值为 Admin，单击 Finish 按钮完成操作。

图 2-6　类创建对话框

③ 创建好的 Admin.java 类的代码如下。

```
package bean;
public class Admin {
}
```

④ 在上面的类里添加如下的属性代码。

```
package bean;
public class Admin {
    private String username;
    private String password;
}
```

⑤ 生成 getters 和 setters。在编辑的 Admin.java 源文件中单击右键，在弹出的菜单中选择 Source/Generate Getters and Setters，如图 2-7 所示。

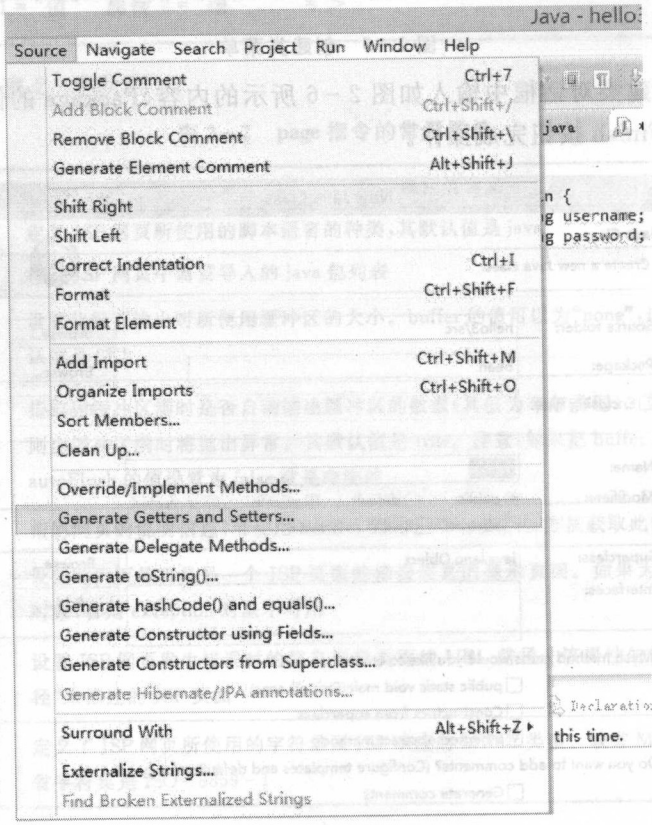


图 2-7 生成 Getters 和 Setters

⑥ 生成后的代码如下。

```
package bean;
public class Admin {
    private String username;
    private String password;
    public String getUsername() {
      returnusername;
    }
    publicvoid setUsername(String username) {
      this.username  = username;
    }
    public String getPassword() {
      returnpassword;
    }
    publicvoid setPassword(String password) {
      this.password  = password;
    }
}
```

⑦ 新建 pageimport.jsp，编辑后的页面代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<%@page import = "bean.Admin" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>导入 Javaben</title>
</head>
<body>
<%
    Admin admin = new Admin("cdap","cdap");
%>
用户名:<% = admin.getUsername() %><br>
密码:<% = admin.getPassword() %><br>
</body>
</html
>
```

代码解释：页面中的代码<%@ pageimport="bean.Admin" %>，显示了如何将一个普通的 Java 类导入到 JSP 页面，然后可以通过普通的 Java 编程来实现对象的创建。

⑧ 运行的结果如图 2-8 所示。

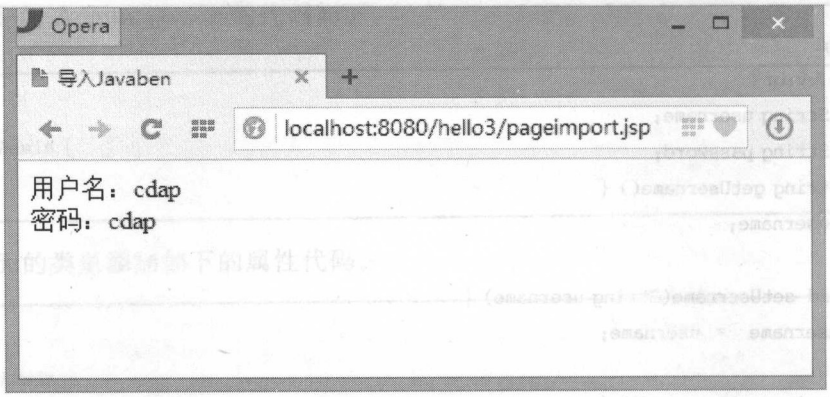


图 2-8　页面 pageimport.jsp 运行结果

2.3.2　include 指令

include 指令功能：指定在 JSP 文件中包含的一个静态的文件，即在 JSP 文件被编译时需要插入的文本或代码。

语法格式如下：

```
<%@include file="文件名"%>
```

当使用 include 指令时，包含文件是静态包含，即这个被包含的文件将被插入到 JSP 文件中去。所包含的文件可以是 JSP 文件、HTML 文件、文本文件、甚至一段 Java 代码。但是在所包含的文件中不能使用“<html></html>”，“<body></body>”标记，因为这将会影响到原有的 JSP 文件中所使用的相同标记。如果所包含的是一个 JSP 文件，则该文件将会被执行。

注意：属性 file 指出了被包含文件的路径，这个路径一般指相对路径，不需要指定端口，协议和域名。

本案例中新建了两个 JSP 页面，其中 date.jsp 页面用于显示系统时间，include.jsp 页面使用 include 指令将 date.jsp 页面包含进来，运行 include.jsp 页面同样可以实现在页面显示系统时间的功能。

① 在工程 hello3 中新建 date.jsp 页面，编辑后的页面代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.text.SimpleDateFormat"%>
<%@page import="java.util.Date" %>
<%
    Date date=new Date();//第 1 行代码
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");//第 2 行代码
    String sdate=sdf.format(date);//第 3 行代码
```

```
%>
当前日期:<% = sdate %>
```

代码解释:第 1 行代码创建日期对象 date;第 2 行代码指定日期的显示格式为年月日,关于日期更多的显示格式可以参看 JDK 帮助文档;第 3 行代码实现将日期对象 date 按照年月日转化成字符串 sdate;<%=sdate %>将格式化的日期显示到页面上。

② 新建 inclue.jsp,编辑页面代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>include file 的使用</title>
</head>
<body>
    <%@include file = "date.jsp" %>
</body>
</html>
```

代码解释:<%@ include file="date.jsp" %>将 date.jsp 页面包含进来,执行本页面可以显示 date.jsp 页面一样的结果。

③ 运行结果如图 2-9 所示。

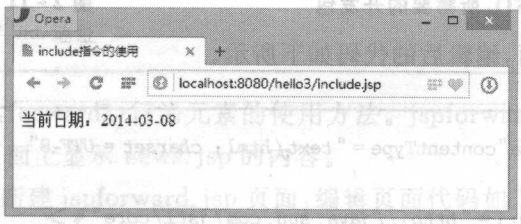


图 2-9　include 指令的运行结果

2.3.3　taglib 指令

taglib 指令允许页面使用标签库。使用标签,可以增加代码的重用程度,比如可以把一些需要迭代显示的内容做成一个标签,在每次需要迭代显示时,就使用这个标签。使用标签也使页面易于维护。

功能:声明 JSP 文件使用了自定义的标签,同时引用标签库,也指定了它们的标签的前缀。

语法格式如下:

```
<%@ taglib uri=" URIToTagLibrary " prefix="tagPrefix" %>
```

属性解释：

● URI：统一资源标记符(Uniform Resource Identifier)，根据标签的前缀对自定义的标签进行唯一的命名。URI 可以是 URL(Uniform Resource Locator)、URN(Uniform Resource Name)或一个路径(相对或绝对)。

● prefix：在自定义标签之前的前缀。

下面将通过一个案例来使用 JSP 提供的 JSTL 标签库。JSTL 是 JSP Standard Tag Library，JSP 标准标签库的缩写。

① 如果要使用 JSTL，需要下面的开发包，如图 2-10 所示。

② 将图 2-10 中的两个文件复制到工程 hello3 中 WEB-INF 的 lib 目录中，复制后如图 2-11所示。

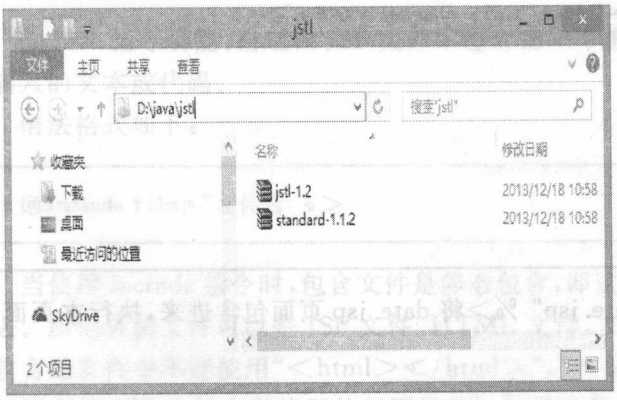


图 2-10　JSTL 所需要的开发包

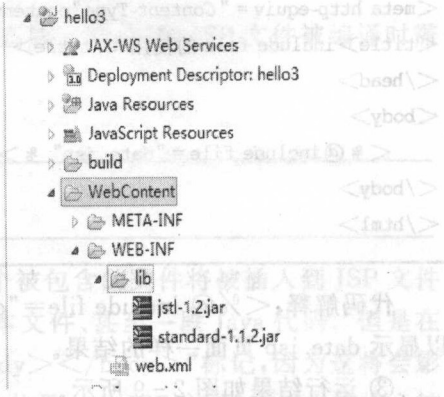


图 2-11　hello3 工程结构图

③ 新建 jstl.jsp 页面，编辑后的代码如下所示。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="c"uri="http://java.sun.com/jstl/core"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>JSTL 的使用</title>
</head>
<body>
    <c:out value="hello jstl"></c:out>
</body>
</html>
```

代码解释：<%@ taglib%>指令声明此 JSP 文件使用了自定义的标签，prefix 定义了在页面里要引用标签时的前缀，uri 用来表示标签描述符。该案例实现用标签输出“hello jstl”。

<c:out>提供了和<% out.println() %>相似的功能，所以运行的结果会在页面上打印出 hello jstl 的字符串，如图 2-12 所示。

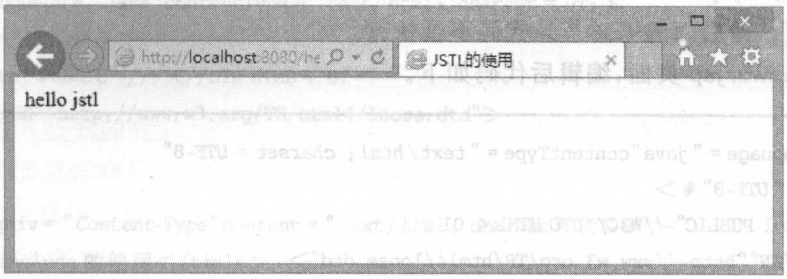


图 2-12　jstl 使用结果

2.4　行为元素

在浏览器请求 JSP 页面时，通常根据行为元素所需要的信息来执行某些动作。行为元素的语法以 XML 为基础，使用时要区分大小写。这里主要讲解 JSP 中常用的 5 项行为元素。

2.4.1　<jsp:forward>

<jsp:forward>用来转移用户的请求，使用户请求的页面从一个页面跳转到另一个页面。此跳转为服务器端的跳转，用户的地址栏不会产生变化。forward 之前的代码会被执行，之后的代码不会被执行。

```
<jsp:forward page = "跳转 URL 地址">
```

本案例介绍了<jsp:forward>行为元素的使用方法。jspforward.jsp 页面转移用户的请求到 news.jsp，从而在页面上显示 news.jsp 的内容。

① 在工程 hello3 中新建 jspforward.jsp 页面，编辑页面代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = utf-8">
<title>JSP forward</title>
</head>
<body>
    <jsp:forwardpage = "news.jsp"></jsp:forward>
```

```
    <% out.print("这行代码不会执行");%>
</body>
</html>
```

代码解释：<jsp:forward>将客户端所发出来的请求从 jspforward.jsp 网页转交给了 news.jsp 网页。而后面的<%%>代码不会执行。

② 新建 news.jsp 页面，编辑后代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>新闻-iOS 7 的完美越狱工具已经到来</title>
</head>
<body>
    <div>各位持有 iOS 终端的用户，好消息来了——iOS 7 的完美越狱工具已经到来。
    据国外媒体消息，大约在苹果向公众发布 iOS 7 之后的第三个月，一款针对这个最新版 iOS 系统的完美越狱工具已经可用了
    </div>
</body>
</html>
```

运行的结果如图 2-13 所示。

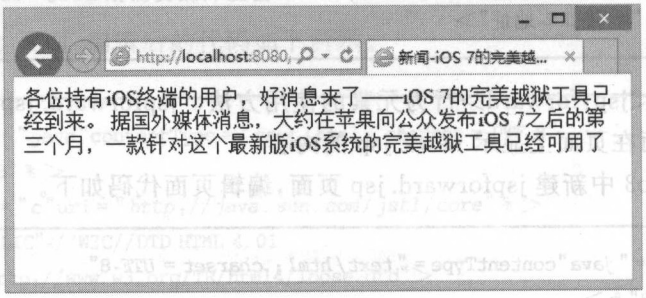


图 2-13　jspforward 页面的运行结果

2.4.2　<jsp:include>

<jsp:include>用来包含静态和动态的文件。语法格式如下：

```
<jsp: include page="URL" flush="true|false"></jsp: include>
```

page 属性用来指定被包含文件的 URL 地址；flush 属性用来指定缓冲区满时，是否进行

清空：当其取值为 true 时，缓冲区满将会被清空；flush 属性默认值为 false。

本案例中使用<jsp:include>行为元素来包含动态文件，用于显示页面当前执行的时间。

① 在工程 hello3 中新建 jspinclude.jsp 页面，其代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>JSP include 的使用</title>
</head>
<body>
    <jsp:includepage="date.jsp"></jsp:include>
</body>
</html>
```

② 运行的结果与图 2-9 相似，区别是显示的时间不同。

注意：在 JSP 中 include 有两种形式，分别是

- <%@ include file="文件名称"%>
- <jsp: include page="URL" flush="true|false"/>

前者是指令元素，后者是动作元素。两者的区别如表 2-3 所列。

表 2-3　Include 指令和<jsp: include>动作的比较

序　号	项　目	include 指令	<jsp: include>动作
1	作用时间	页面转化期间	请求期间
2	包含内容	文件的实际内容	页面的输出
3	影响主页面	可以	不可以
4	内容变化是否需要手动修改包含页面	需要	不需要
5	编译速度	较慢(资源必须被解析)	较快
6	执行速度	较快	较慢(每次资源必须被解析)
7	灵活性	较差(页面名称固定)	更好(页面可以动态指定)

介于<jsp:include>动作在维护上的优势，对于文件包含，应该尽可能地使用 include 动作。仅在所包含的文件中定义了主页面要用到的字段或方法，或所包含的文件设置了主页面的响应报头时，才应该使用 include 指令。

2.4.3　<jsp:useBean>

<jsp:useBean>行为元素的作用是使一个 JavaBean 组件在该页中可用。使用这个动作，JSP 可以动态使用 JavaBean 组件来扩充 JSP 的功能。

<jsp:useBean>首先会尝试定位 Bean 实例，在 scope 属性指定的作用域使用指定的名称(id 属性值)定位 Bean 对象，如果其不存在，则依据 class 名称(class 属性指定)或序列化模板(beanName 属性指定)进行实例化。

本案例中使用到的语法格式如下。

```
<jsp:useBean id = "name" scope = " page|request|session|application " class = " className "/>
```

id="name"，一个用来标识定位作用域的变量。可以在 JSP 文件的表达式或脚本小应用程序中使用该变量名称。该名称大小写敏感，必须符合 JSP 页面中脚本语言的命名规则。假如使用的是 Java 语言，则该名称遵守的是 Java 命名规范。假如该 Bean 对象已由其他<jsp:useBean>元素创建，则该值必须和实例化该 Bean 对象的<jsp:useBean>元素 id 属性值一致，才能实现定位到该 Bean 对象。

scope="page | request | session | application"，表示 Bean 对象存在的作用范围，默认值为 page。不同作用域解释如下：

① page：<jsp:useBean>元素所在 JSP 页面或其静态包含页面使用该 JavaBean 对象，直到该页面发送响应回客户端或跳转(forwards)至其他页面。

② request：可以在处理同一个请求的任意一个页面使用该 JavaBean 对象，直到该页面发送响应回客户端或产生新的请求。

③ session：可以在同一次会话的任意一个页面使用该 JavaBean 对象，该 JavaBean 对象在整个会话期间一直存在。使用<jsp:useBean/>创建 JavaBean 对象的页面的<% @ page %>指令元素的 session 属性值必须设置为 true。

④ application：用户可以在创建该 JavaBean 对象的同一个应用的任意一个页面使用该 JavaBean 对象，该 JavaBean 对象在整个应用期间一直存在。应用中任意一个页面均可使用它。

class="package. class"。从一个 class 实例化 Bean 对象，使用 new 关键字调用 class 的构造方法完成。该 class 必须不能是抽象，必须有一个 public、无参的构造器。注意包的名字需区分大小写。

2.4.4 <jsp:setProperty>

<jsp:setProperty>行为用于设置一个或多个 bean 属性的值。使用语法如下：

```
<jsp:setProperty name = "beanName" property = "propertyName"
    [param = "parameterName"|value = "value"] />
```

如果 property 的属性值为 *，表示全部的属性，* 是一个通配符。

2.4.5 <jsp:getProperty>

<jsp:getProperty>操作是对<jsp:setProperty>操作的补充，它用来访问一个 Bean 的属性。它访问的属性值将它转化成一个 String，然后发送到输出流中。如果属性是一个对象，

将调用 toString()方法。它的使用方法如下：

```
<jsp:getProperty name = "beanName" property = "propertyName" />
```

本案例主要介绍<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>的使用方法。

① 在工程 hello3 中创建 input.jsp，编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>JSP useBean 使用</title>
</head>
<body>
    <form action = "jspusebean.jsp"method = "post">
    <input type = "text"name = "username">
    <input type = "text"name = "password">
    <input type = "submit"value = "提交">
    </form>
</body>
</html>
```

② 创建 jspusebean.jsp，编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<jsp:useBeanid = "admin" scope = "page"class = "bean.Admin"/>
<jsp:setPropertyname = "admin" property = " * "/>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = utf-8">
<title>jsp Usebean</title>
</head>
<body>
你输入的用户名和密码为:<br>
用户名:<jsp:getPropertyname = "admin"property = "username"/><br>
密码:<jsp:getPropertyname = "admin"property = "password"/><br>
```

```
</body>
</html>
```

代码解释：<jsp:useBean></jsp:useBean>代码定义了一个 JavaBean，其中 id 为 admin，值的范围为 page，用于实例化 admin 对象类为：bean. Admin。<jsp:setProperty><jsp:setProperty>代码将 input. jsp 页面中输入的属性值赋给 admin 的属性，其中 * 代表全部赋值，赋值的规则是查找<input>标签中的 name 值和 admin 对象中属性值相同的。如果相同，就赋值给 admin 对应的属性值。<jsp:getProperty></jsp:getProperty>获取 admin 对象中对应属性的值。

③ 运行的结果如图 2-14 和图 2-15 所示。

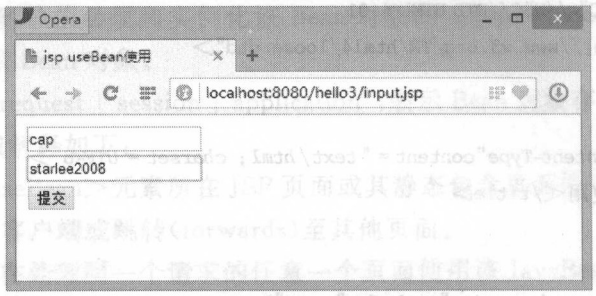


图 2-14　input. jsp 的运行结果

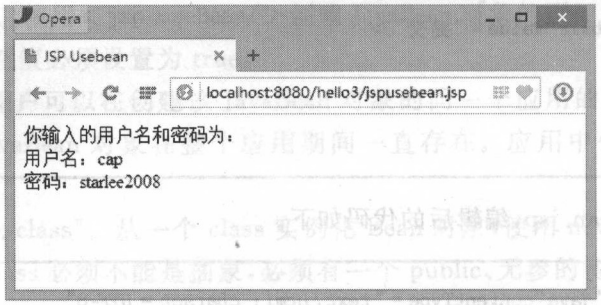


图 2-15　数据提交后 jspusebean 运行的结果

第3章 JSP 对象

3.1 out 内置对象

JSP 对象包括内置对象和非内置对象(需通过 new 关键字创建的对象),内置对象是可以不加声明和创建就可以在 JSP 页面脚本中便用的成员变量,常用的内置对象包括 out 对象、request 对象、response 对象、session 对象、page 对象、application 对象、exception 对象和 config 对象等。本章主要讲解前 4 种内置对象以及 Cookie 对象,并给出详细的实例。

3.1.1 out 对象概念

out 对象能把结果输出到网页上,主要是用来控制管理输出的缓冲区(buffer)和输出流(output stream)。out 对象用于各种数据的输出,其常用方法及描述如表 3-1 所列。out 对象被封装为 javax. servlet. jsp. JspWriter 接口,通过调用 pageContext. getOut()方法可以获取 out 对象。

表 3-1 列出了 out 内置对象常用的方法,下面将通过案例来讲解 out 内置对象的使用。

表 3-1 out 对象的常用方法

序 号	方 法	描 述
1	public abstract void clear()	清除缓冲区中的内容,不将数据发送至客户端
2	public abstract void clearBuffer()	将数据发送至客户端后,清除缓冲区中的内容
3	public abstarct void close()	关闭输出流
4	public abstract void flush()	输出缓冲区中的数据
5	public int getBufferSize()	取缓冲区的大小。缓冲区的大小可用<%@ page buffer="size" %>设置
6	public abstract intgetRemainning()	获取缓冲区剩余空间的大小
7	public boolean isAutoFlush()	获取用<%@ page is AutoFlush="true/false"%>设置的 AutoFlush 值
8	public abstract void newLine()	输出一个换行字符,换一行
9	public abstract void print()	显示各种数据类型的内容
10	public abstract void println()	分行显示各种数据类型的内容

3.1.2 out 对象使用案例

本案例主要讲解 out 内置对象的 print 方法,print 方法可以输出不同类型的数据,包括字

符数组，双精度浮点数，字符串等类型。

新建 Dynamic Web Project 工程，工程名为 hello4，在 WebContent 下新建 out.jsp 文件，编辑 out.jsp 的代码如下：

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>out demo</title>
</head>
<body>
<%
    out.print("输出字符数组<br>");
    out.print(newchar[]{'c','h','a','r'});
    out.print("输出双精度数<br>");
    out.print(12.4);
    out.print("输出单精度数<br>");
    out.print(12.4f);
%>
</body>
</html>
```

读者可以试试表 3-1 中 out 内置对象常用方法。

运行的结果如图 3-1 所示。

图 3-1　out.jsp 运行结果

3.2　request 内置对象

3.2.1　request 对象概念

request 内置对象封装客户端的请求信息，通过它可以了解到客户的需求(如：请求的来源、http 头部、cookies 和请求相关的参数值等)，然后做出响应。它是 HttpServletRequest 类的实例。request 内置对象常用的方法如表 3-2 所列。

表 3-2　request 内置对象常用方法

序　号	方　法	描　述
1	object getAttribute(String name)	返回指定属性的属性值
2	String getCharacterEncoding()	返回字符编码方式
4	ServletInputStream getInputStream()	得到请求体中一行的二进制流
5	String getParameter(String name)	返回 name 指定参数的参数值
6	Enumeration getParameterNames()	返回可用参数名的枚举
7	String[] getParameterValues(String name)	返回包含参数 name 的所有值的数组
8	String getProtocol()	返回请求用的协议类型及版本号
9	String getScheme()	返回请求用的计划名，如：http、https 及 ftp 等
10	String getServerName()	返回接受请求的服务器主机名
11	int getServerPort()	返回服务器接受此请求所用的端口号
12	String getRemoteAddr()	返回发送此请求的客户端 IP 地址
13	String getRemoteHost()	返回发送此请求的客户端主机名
14	void setAttribute(String key,Object obj)	设置属性的属性值

3.2.2　request 使用案例

本案例介绍 request 内置对象的使用方法，包含解决中文乱码的办法。

request.jsp 页面的编码是 UTF-8，Java 默认的编码是 Unicode，需要在处理接收参数前进行编码设置，以解决中文乱码问题。

① 继续在 hello4 的工程目录“WebContent”下新建 request.jsp 页面，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>提交中文信息</title>
</head>
```

```
<body>
    <form action = "handle1.jsp" method = "post">
    <input type = "text" name = "str">
    <input type = "submit" value = "提交">
    </form>
</body>
</html>
```

② 新建 handle1.jsp 页面，编辑的代码如下。

```
<%@page language = "java" contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type" content = "text/html; charset = UTF-8">
<title>处理中文</title>
</head>
<body>
<%
    //第一步，提交数据的网页的字符编码为 utf-8
    request.setCharacterEncoding("utf-8");//第 1 行代码
    String str = new String(request.getParameter("str"));//第二步设置编码  第 2 行代码
%>
<% = str %>//第 3 行代码
</body>
</html>
```

代码解释：第 1 行代码(request.setCharacterEncoding("utf-8");)设置请求的编码格式为 utf-8。第 2 行代码(String str=new String(request.getParameter("str"));)表获得表单传递的数据，并使用 String 的构造函数将字符转为 utf-8。第 3 行代码(<%=str %>)表将转化后的代码通过表达式显示到页面。

③ 运行 request.jsp，如图 3-2 所示输入“成都航空职业技术学院”，单击“提交”。运行结果如图 3-3 所示。

图 3-2　request.jsp 运行的结果

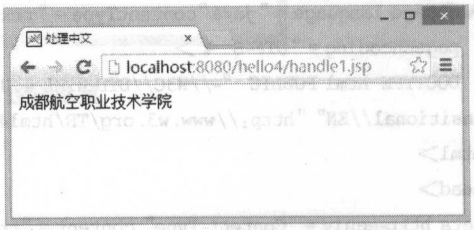


图 3-3　handle1.jsp 的处理结果

3.3　response 内置对象

3.3.1　response 对象概念

response 对象包含了响应客户请求的有关信息，主要将 JSP 处理数据后的结果传回到客户端。response 对象实现使用 javax. servlet. http. HttpServletResponse 接口，它是 Http ServletResponse 类的实例。response 对象所提供的常用方法如表 3 - 3 所列。

表 3 - 3　response 内置对象常用方法

序　号	方　法	描　述
1	String getCharacterEncoding()	返回响应用的是何种字符编码
2	ServletOutputStream getOutputStream()	返回响应的一个二进制输出流
3	PrintWriter getWriter()	返回可以向客户端输出字符的一个对象
4	void setContentLength(int len)	设置响应头长度
5	void setContentType(String type)	设置响应的 MIME 类型
6	sendRedirect(java. lang. String location)	重新定向客户端的请求
7	void setHeader(String name, String value)	指定 String 类型的值到名为 name 的 http 头部

3.3.2　response 对象使用案例

本案例主要是使用 response 对象的 sendRedirect 方法来实现页面的重定向。response-demo. jsp 页面设置了一个下拉选项框控件，通过下拉选值向 handle. jsp 页面提交数据。handle. jsp 根据提交的数据重定向网页的内容。

① 新建 Dynamic Web Project 工程，工程名为 hello4，在 WebContent 下新建 response1. jsp 页面，实现网页的重定向。编辑的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>response demo</title>
</head>
<body>
    <form action="handle2.jsp"method="post">
    <select name="opt">
    <option value="sina">新浪</option>
    <option value="sohu">搜狐</option>
    <option value="163">网易</option>
```

```
    </select>
    <input type = "submit"value = "go">
    </form>
</body>
</html>
```

② 接着创建处理页面 handle2.jsp,编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>response 内置对象的使用</title>
</head>
<body>
<%
    String opt = request.getParameter("opt");//第 1 行代码
    if(opt.equals("sina"))
    {
        response.sendRedirect("http://www.sina.com.cn");
    }elseif(opt.equals("sohu"))
    {
        response.sendRedirect("http://www.sohu.com");
    }else
        response.sendRedirect("http://www.163.com");

%>
</body>
</html>
```

代码解释:第 1 行代码使用 request 内置对象获得表单提交的数据,余下的代码使用了 if else 判断获得的数据,如果是指定的字符串,则跳转到指定的门户网站。

③ 运行 responsel.jsp 的结果如图 3-4 所示,当单击 go 会根据用户选项跳转到指定的网页。

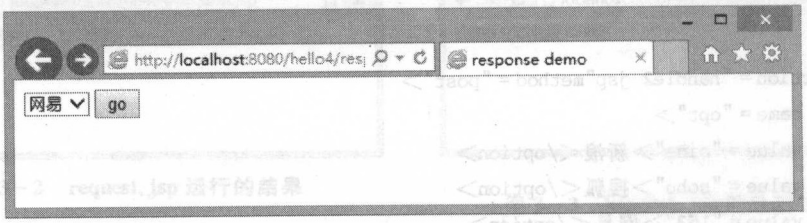


图 3-4 responsedemo.jsp 页面运行结果

下面这个案例将展示 response 对象的其他方法，用以实现页面刷新。

① 在 WebContent 下新建 response2.jsp 页面，实现页面的刷新。编辑的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>refresh</title>
</head>
<body>
<%
    response.setHeader("refresh", "1");//第 1 行代码
    out.print((new Date()).toString());//第 2 行代码

%>
</body>
```

代码解释：第 1 行代码(response.setHeader("refresh", "1");)通过 response 设置头的刷新时间为 1 秒，第 2 行代码(out.print((new Date()).toString());)实现将日期字符串通过 out 输出到页面。

② 运行的结果如图 3-5 所示。

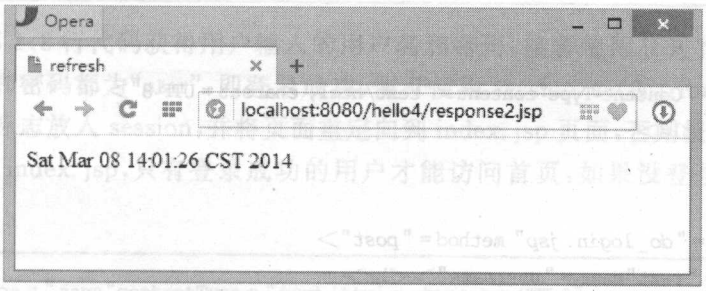


图 3-5　responsedemo1.jsp 运行结果

3.4　session 内置对象

3.4.1　session 对象概念

session 对象在第一个 JSP 页面被装载时自动创建，从客户端连到服务器开始，直到客户端与服务器断开连接为止，被称为一个会话。表 3-4 所列为 session 内置对象常用方法。session 是 javax.servlet.http.HttpSession 接口的实例化对象。

表 3-4 session 内置对象常用方法

序 号	方 法	描 述
1	long getCreationTime()	返回 session 创建时间
2	public String getId()	返回 session 创建时 JSP 引擎为它设的唯一 ID 号
3	long getLastAccessedTime()	返回此 session 里客户端最近一次请求时间
4	int getMaxInactiveInterval()	返回两次请求间隔多长时间此 session 被取消
5	String getValueNames()	返回一个包含此 session 中所有可用属性的数组
6	void invalidate()	取消 session,使 session 不可用
7	boolean isNew()	返回服务器创建的一个 session,客户端是否已经加入
8	VoidremoveValue(String name)	删除 session 中指定的属性
9	void setMaxInactiveInterval()	设置两次请求间隔多长时间此 session 被取消
10	setAttribute(String key, Object value)	使用指定的名称和值来产生一个对象并绑定到 session 中
11	getAttribute(String key)	返回 session 对象中与指定名称绑定的对象,如果不存在则返回 null

3.4.2 response 对象使用案例

本案例主要通过 session 实现登录判断。

① 在工程 hello4 下新建 login.jsp 登录页面,编辑的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>登录</title>
</head>
<body>
    <form action="do_login.jsp" method="post">
    <input type="text"name="username"><br>
    <input type="text"name="password"><br>
    <input type="submit"value="登录">
    </form>
</body>
</html>
```

② 接着是新建登录的业务处理页面 do_login.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
```

```
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>handle login</title>
</head>
<body>
<%
    String name = request.getParameter("username");
    String pass = request.getParameter("password");
    if(name! = null&&pass! = null)
    {
        if(name.equals("star")&&pass.equals("star"))
        {
            session.setAttribute("u", name);
            session.setAttribute("flag", "ok");
            response.sendRedirect("index.jsp");
        }else
        {
            out.print("用户名或者密码不对,请重新输入");
        }
    }
%>
</body>
</html>
```

代码解释:第 1、2 行代码获得用户输入的用户名和密码,接着使用 if 进行判断,如果客户端输入的用户名和密码都为“star”,即登录成功,便可使用 session.setAttribute()方法将用户名及验证成功的标志放入 session,并将页面重定向到 index.jsp 页面,否则报错。

③ 新建首页 index.jsp,只有登录成功的用户才能访问首页,如果没登录成功,会跳转到登录页面。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>欢迎</title>
</head>
<body>
<%
```

```
    String flag = (String)session.getAttribute("flag");//第 1 行代码
    if(flag!  = null){
        String name = (String)session.getAttribute("u");
        out.print("欢迎" + name + "的到来");
    }else
    {
        out.print("你还没登录");
        response.sendRedirect("login.jsp");}
     %>
</body>
</html>
```

代码解释:第 1 行代码获得 session 中的 flag 值,接着判断 flag 值,如果不为空,则输出欢迎信息,否则提示还没登录,并跳转到登录页面。

④ 运行本案例,如图 3-6 输入用户名和密码进行登录,登录成功的结果如图 3-7 所示,直接转到 index.jsp 首页了。

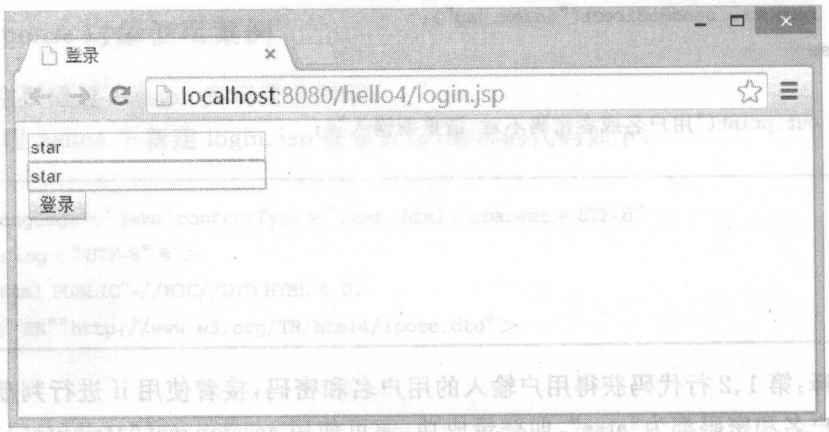


图 3-6　login.jsp 的运行界面

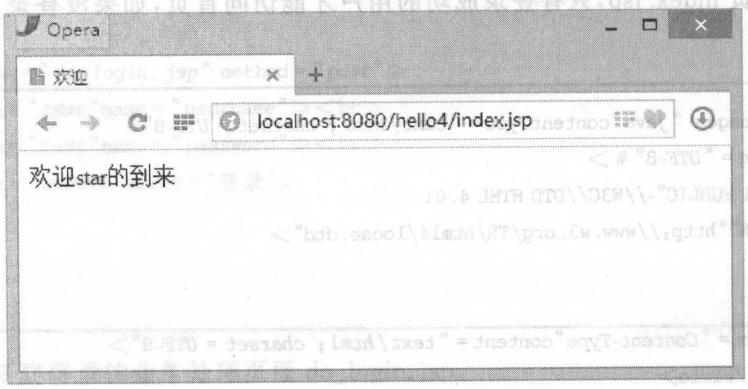


图 3-7　登录成功运行结果

3.5　Cookie 对象

3.5.1　Cookies 对象概念

浏览器和 Web 服务器使用 HTTP 协议通信，当某用户发出页面请求，Web 服务器只进行简单响应，就关闭与该用户的连接。对于用户请求无论是否第一次提出，服务器都会作为第一次来处理，显然不利于提高程序运行效率。为了解决这个问题(bug)，Netscape 开发出 Cookie 这个有效的工具用以保存识别用户信息。Cookie 是一种 Web 服务器借助浏览器可在访问者硬盘上存取信息的技术。Cookie 具有以下优势：

① Cookie 能跟踪统计特定访问者的访问次数、最后访问时间和访问者进入站点的路径。

② Cookie 能统计网站上各类的广告点击量，帮助网站经营者决策投放收效更好的广告，从而可以更精确地投放广告。

③ Cookie 可以方便用户在有效期未到时不输入密码和用户名就可浏览站点的其他页面。

④ Cookie 能帮助站点统计用户个人资料以实现各种各样的个性化服务。

3.5.2　创建 Cookie

创建 Cookie 语法格式为：

```
<%
    Cookie username_Cookie = new Cookie("username", " starlee2008");
    cookie.setMaxAge(10);
    response.addCookie(username_Cookie);
%>
```

说明：

JSP 是调用 Cookie 对象相应的构造函数 Cookie(name,value)，用合适的名字和值来创建 Cookie。初始化有两个参数，第一个参数 cookiename 定义了 Cookie 的名字，后一个参数 value 是一个字符串，定义了 Cookie 的内容，也就是用户希望网页在用户的机器上标识的文件内容。

cookie.setMaxAge(10)，调用了 Cookie 中的 setMaxAge 方法，设定 Cookie 在用户机器硬盘上的存活期为 10 秒。一个 Cookie 在用户的硬盘里面存在的时间并不是无限期的，在建立 Cookie 对象的时候，必须指定 Cookie 的存活期。超过了这个存活期后，Cookie 文件就不再起作用，会被用户的浏览器自行删除。如果希望用户在下次访问这个页面的时候，Cookie 文件仍然有效而且可以被网页读出来的话，可以将 Cookie 的存活期设置稍微长一些。例如 cookie.setMaxAge(365 * 24 * 60 * 60)可以让 Cookie 文件在一年内有效。

在 JSP 中，程序是通过 cookie.setXXX 设置各种属性，用 cookie.getXXX 读出 cookie 的属性，Cookie 对象的常用方法如表 3-5 所列。

表 3-5 Cookie 对象的常用方法

序 号	方 法	描 述
1	int getMaxAge()	返回 Cookie 过期之前的最大时间，以秒计算
2	String getName()	返回 Cookie 的名字
3	String getPath()	返回 Cookie 适用的路径。如果不指定路径，Cookie 将返回给当前页面所在目录及其子目录下所有页面
4	boolean getSecure()	如果浏览器通过安全协议发送 cookies 将返回 true 值，如果浏览器使用标准协议则返回 false 值
5	int getVersion()	返回 Cookie 所遵从的协议版本
6	void setDomain(String pattern)	设置 cookie 中 Cookie 适用的域名
7	void setMaxAge(int expiry)	以秒计算，设置 Cookie 过期时间
8	void setSecure(boolean flag)	指出浏览器使用的安全协议，例如 HTTPS 或 SSL
9	void setValue(String newValue)	cookie 创建后设置一个新的值

3.5.3 读取 Cookie

Cookie 文件创建好后，JSP 将调用 request. getCookies()从客户端读入 Cookie，getCookies()方法返回一个 HTTP 请求头中的内容对应的 Cookie 对象数组。用循环访问该数组的各个元素，调用 getName 方法检查各个 Cookie 的名字，直至找到目标 Cookie。然后对该 Cookie 调用 getValue 方法取得与指定名字关联的值，如下面的代码片段所示。

```
<%
    String name = request.getParameter("username");
    Cookie c = new Cookie("user",name);
    c.setMaxAge(60);
    response.addCookie(c);
    response.sendRedirect("readcookie.jsp");
%>
<%
    Cookie c = null;
    String name = null;
    Cookie cookies[] = request.getCookies();//获得所有的 cookie
    if(cookies! = null)
    {
    for(int i = 0;i<cookies.length;i + + )
    {
    c = cookies[i];
    if(c.getName().equals("user"))
    {
    name = c.getValue();
    }
```

```
    }
    if(name! =null)
    out.print(name + "有效期为 10s");
    else
    out.print("cookie 已经失效");
}
%>
```

3.5.4　Cookie 使用案例

本案例中 cookie.jsp 页面向 handle.jsp 页面提交信息，在 handle.jsp 页面获取提交的信息，创建 cookie 对象，并通过 response.addCookie 方法发送一个 HTTP Header，重定向到 readcookie.jsp 页面，读取存储的 cookie。cookie 有效期设置为 10 秒，如果超过 10 秒读取的话页面上会显示读取 cookie 已经失效的信息。

① 在工程 hello5 中，新建 JSP 文件 cookie.jsp，编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>cookie</title>
</head>
<body>
    <form action = "handle_cookie.jsp" method = "post">
    <input type = "text"name = "username">
    <input type = "submit"value = "提交">
    </form>
</body>
</html>
```

② 建表单处理页面 handle_cookie.jsp，编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>handle</title>
```

```
</head>
<body>
<%
    String name = request.getParameter("username");//第 1 行代码
    Cookie c = new Cookie("user",name);//第 2 行代码
    c.setMaxAge(10);//第 3 行代码
    response.addCookie(c);//第 4 行代码
    response.sendRedirect("readcookie.jsp");//第 5 行代码
%>
</body>
</html>
```

代码解释:第 1 行代码获得用户提交的用户名,第 2 行代码创建 Cookie 对象 c,第 3 行代码设置 Cookie 的有效期为 10 s,第 4 行代码将 Cookie 对象 c 添加到 response,第 5 行重定向到页面 readcookie.jsp。

③ 最后创建读取 cookie 的 readcookie.jsp 页面。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>read cookie  有效期为 10s</title>
</head>
<body>
<%
    Cookie c = null;//第 1 行代码
    String name = null;//第 2 行代码
    Cookie cookies[] = request.getCookies();//获得所有的 cookie  //第 3 行代码
    if(cookies! = null)
    {
        for(int i = 0;i<cookies.length;i + + )
        {
            c = cookies[i];
            if(c.getName().equals("user"))
            {
                name = c.getValue();
            }
        }
        if(name! = null)
            out.print(name + "有效期为 10s");
        else
```

```
        out.print("cookie 已经失效");
}
%>
</body>
</html>
```

代码解释：第 1、2 行代码定义了 Cookie 对象 c 和 String 对象 name。第 3 行代码获得所有的 Cookie 对象数组。余下的代码通过 if 判断，如果 cookies 数组不为 null，就进一步查找定义为 user 的 Cookie，如果找到，并在 10 s 有效期内，则在页面输出用户名，否则打印出 cookie 已经失效。

④ 在如图 3－8 所示界面上单击“提交”按钮，成功读取 cookie 的结果如图 3－9 所示，读取 cookie 失效的结果如图 3－10 所示。

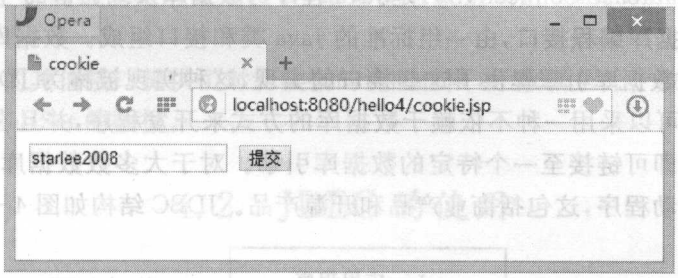


图 3－8　cookie.jsp 的运行结果

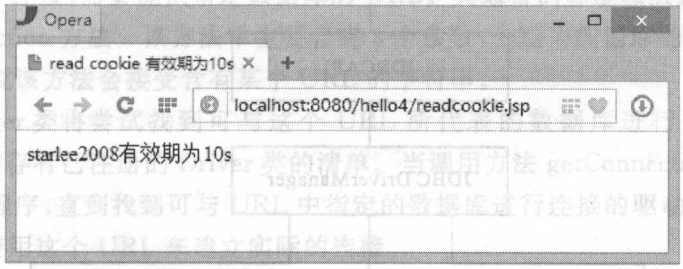


图 3－9　成功读取到 cookie 的页面

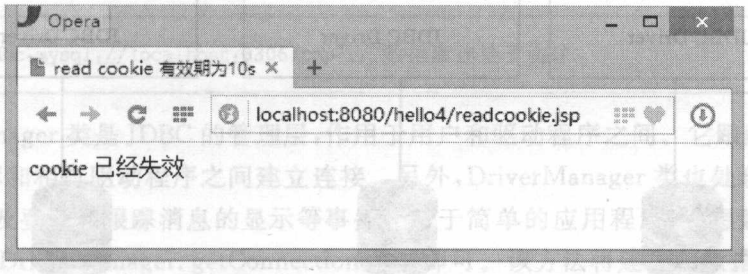


图 3－10　读取 cookie 失效的结果

第4章

JDBC 数据库访问技术

4.1 JDBC 的概念

JDBC (Java Database Connectivity)是 Java 程序与数据库系统通信的一种规范,是一套标准 API,即 Java 数据库编程接口,由一组标准的 Java 类和接口组成。数据库引擎开发商和第三方厂商针对特定数据库引擎提供了这些接口的实现,这种实现被称为 JDBC 驱动程序(JDBC Driver)。由此可以采用一种不依赖于数据库的方式来开发程序,并且在部署时适当地插入 JDBC 驱动程序即可链接至一个特定的数据库引擎。对于大多数数据库引擎,市场上都有了相应的 JDBC 驱动程序,这包括商业产品和开源产品。JDBC 结构如图 4-1 所示。

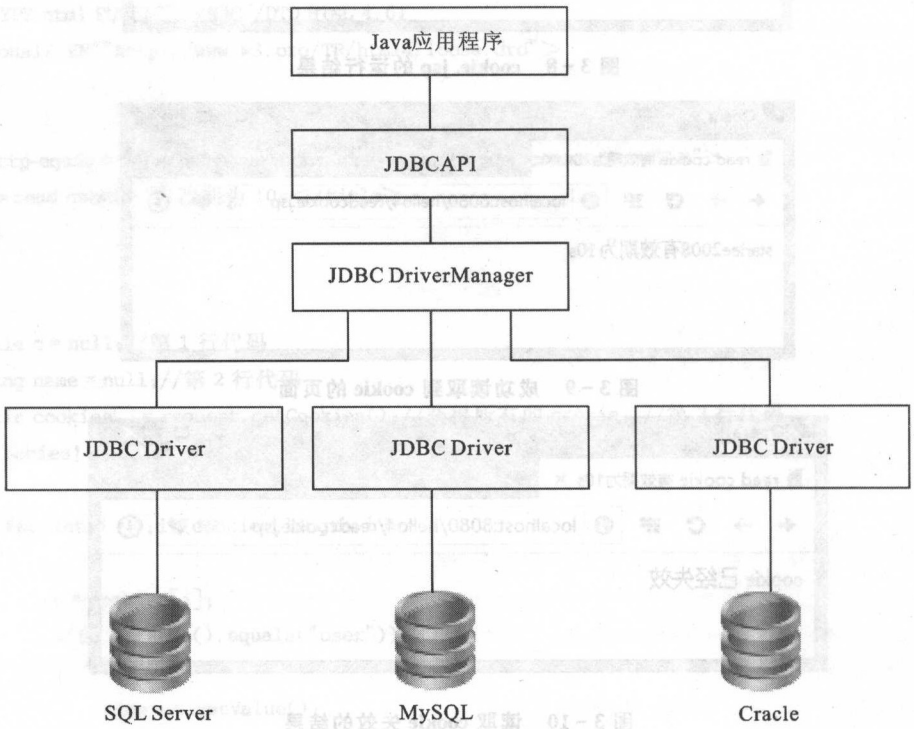


图 4-1 JDBC 结构图

DriverManger 是所有其他接口实现的切入点。当载入 Driver 时,它将用 DriverManger 注册自身。当 JDBC 应用程序需要和某个数据库相连接时,它会向 DriverManger 要求一个连

接，然后 DriverManger 会询问每个注册的 Driver，问它是否知道如何为请求的数据库创建连接。当某个 Driver 回答“是”的时候，DriverManger 会代表应用程序向它要求一个 Connection，然后该 Driver 试图建立一个连接，并返回给应用程序。

Connection 是另一种核心 JDBC 类型。通过 Connection 实例，JDBC 应用程序可以创建不同类型的 Statement 实例。主要的 Statement 类型可以执行一个简单的 SQL 语句，例如 select、update 或 delete，当执行 select 语句时，结果会作为 ResultSet 的实例被返回。ResultSet 有一些方法，用这些方法可以浏览结果行，并请求当前行中的列值。

CallableStatement 是用于存储过程的。与用于 PreparedStatement 的一样，也可以为输入参数赋值，除此以外，还有一些用于声明输出参数类型的方法。

Statement 用于执行静态 SQL 语句并返回它所生成结果的对象接口，通过 Connection 中的 createStatement 方法得到。

Resultset 用于指向结果集对象的接口，结果集对象是通过 Statement 中的 execute 等方法得到的。

JDBC API 中的其他接口提供了对元数据的访问权限，元数据是关于数据库和 JDBC 驱动程序本身以及 ResultSet 的数据。

4.2　JDBC 的使用

1. DriverManger 驱动加载

要实现 JDBC 连接，需要提供指定数据库的 URL。其通常的标准做法是调用 DriverManager 的 getConnection 方法。该方法中主要含有 3 个参数，分别为数据库的 URL、连接用户名和密码。也就是说该方法会接受含有某个 URL 的字符串。

DriverManager 类将尝试找到可与这个 URL 所代表的数据库进行连接的驱动程序。DriverManager 类存有已注册的 Driver 类的清单。当调用方法 getConnection 时，它将检查清单中的每个驱动程序，直到找到可与 URL 中指定的数据库进行连接的驱动程序为止。Driver 的方法 connect 使用这个 URL 来建立实际的连接。

下述代码显示了一个 JDBC URL。

```
String url = "jdbc:mysql://localhost:3306/cap";//数据库连接子协议
```

DriverManager 类是 JDBC 的管理层，作用于用户和驱动程序之间。它跟踪可用的驱动程序，并在数据库和相应驱动程序之间建立连接。另外，DriverManager 类也处理诸如驱动程序登录时间限制及登录和跟踪消息的显示等事务。对于简单的应用程序，要想建立与数据库的连接只需使用 DriverManager. getConnection 方法即可。该方法将建立与数据库的连接。

DriverManager 中注册驱动的方式如下：

(1) 调用方法 Class. forName

调用方法 Class. forName 可显式地加载驱动程序类。由于这与外部设置无关，因此推荐使用这种加载驱动程序的方法。加载类的代码为

```
Class.forName("com.mysql.jdbc.Driver");//加载数据库驱动
String url = "jdbc:mysql://localhost:3306/cap";//数据库连接子协议
```

(2) 将驱动程序添加到 java.lang.System 的属性 jdbc.drivers 中

将驱动程序添加到 java.lang.System 的属性 jdbc.drivers 中是一个由 DriverManager 类加载的驱动程序类名的列表,由冒号":"分隔。初始化 DriverManager 类时,它搜索系统属性 jdbc.drivers,则 DriverManager 类将试图加载它们。

```
<%
//系统属性指定数据库驱动列表
System.setProperty("jdbc.driver","com.mysql.jdbc.Driver:oracle.jdbc.driver.OracleDriver ");
//数据库连接子协议
String url = "jdbc:mysql://localhost:3306/cap";
%>
```

(3) 通过创建驱动对象来加载数据库驱动

下面代码中的关键语句"new com.mysql.jdbc.Driver()"可实现创建 driver 对象,加载数据库驱动。

```
new com.mysql.jdbc.Driver();//创建 driver 对象,加载数据库驱动
String url = "jdbc:mysql://localhost:3306/cap";//数据库连接子协议
```

2. 建立连接

加载 Driver 类,并在 DriverManager 类中注册后,它们即可用来与数据库建立连接。当调用 DriverManager.getConnection 方法发出连接请求时,DriverManager 将检查每个驱动程序,查看它是否可以建立连接。有时会有多个不同类型的 JDBC 驱动程序可以与给定的 URL 连接。此情况下,DriverManager 将轮流在每个驱动程序上调用方法 Driver.connect,并向其传递最开始用户在 DriverManager.getConnection 方法中传递的 URL,然后逐一对驱动程序进行测试,选择出第一个可以成功连接到指定 URL 的驱动程序。当然创建数据库连接除了提供数据库的 URL 和驱动类型,还需提供访问数据库的用户名和密码。这种方法最初看起来效率不高,但由于不可能同时加载数十个驱动程序,因此每次连接实际只需几个过程调用和字符串比较。

3. 发送 SQL 语句

Statement 对象用于将 SQL 语句发送到数据库中。实际上有三种 Statement 对象,它们都作为在给定连接上执行 SQL 语句的包容器:Statement、PreparedStatement(从 Statement 继承而来)和 CallableStatement(从 PreparedStatement 继承而来)。它们都专用于发送特定类型的 SQL 语句:Statement 对象用于执行不带参数的简单 SQL 语句;PreparedStatement 对象用于执行带或不带 IN 参数的预编译 SQL 语句;CallableStatement 对象用于执行对数据库已存储过程的调用。Statement 接口提供了执行语句和获取结果的基本方法;PreparedState-

ment 接口添加了处理 IN 参数的方法；而 CallableStatement 添加了处理 OUT 参数的方法。

创建 Statement 对象如下。

建立了到特定数据库的连接之后，就可用该连接发送 SQL 语句了。Statement 对象是用 Connection 的方法 createStatement 创建，如下列代码片段中所示。

```
Connection conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);
String sql = "select * from admin order by id";
Statement stmt = conn.createStatement();
```

为了执行 Statement 对象，被发送到数据库的 SQL 语句将被作为参数提供给 Statement 的方法，如下代码所示。

```
ResultSet rs = stmt.executeQuery(sql);
```

4. 使用执行语句

Statement 接口提供了三种执行 SQL 语句的方法：executeQuery、executeUpdate 和 execute。使用哪一个方法由 SQL 语句所产生的内容决定。

execute 方法用来执行任意的 SQL 查询，如果查询的结果是一个 ResultSet，这个方法就返回 true。如果结果不是 ResultSet，比如是 insert 或者 update 查询，它就会返回 false。可以通过它的 getResultSet 方法来获取 ResultSet，或者通过 getUpdateCount()方法来获取更新的记录条数。

executeQuery 方法用于产生单个结果集的语句，例如 SELECT 语句。

executeUpdate 方法用于执行 INSERT、UPDATE 或 DELETE 语句以及 SQL DDL(数据定义语言)语句，例如 CREATE TABLE 和 DROP TABLE。INSERT、UPDATE 或 DELETE 语句的效果是修改表中零行或多行中的一列或多列。executeUpdate 的返回值是一个整数，表示受影响的行数(即更新计数)。对于 CREATE TABLE 或 DROP TABLE 等不操作行的语句，executeUpdate 的返回值为零。

5. 语句完成

当连接处于自动提交模式时，其中所执行的语句在完成时将自动提交或还原。语句在已执行且所有结果返回时，即认为已完成。对于返回一个结果集的 executeQuery 方法，在检索完 ResultSet 对象的所有行时该语句则完成。对于方法 executeUpdate，当它执行时语句即完成。但在少数调用方法 execute 的情况中，在检索所有结果集或它生成的更新计数之后语句才完成。

6. 关闭对象

Statement 对象将由 Java 垃圾收集程序自动关闭。而良好的编程风格应在不需要 Statement 对象时显式地关闭它们。这将立即释放 DBMS 资源，有助于避免潜在的内存问题。

本章以及后面的章节使用到的数据库为 MySQL5。安装 MySQL 数据库服务器后，运行随书附带的 hell_inint 工程可以方便读者创建运行本书案例的初始环境。MySQL 的管理工具为 Navicat，Navicat 创建数据库连接，建立表，存储过程的使用详见附录 B。

4.3 脚本方式进行数据库连接

本案例主要讲解使用 mysql - connector - java 驱动来连接 MySQL 数据库，如果连接成功则页面中会有成功的提示，反之亦然。在 MySQL 数据库中已经建立了 cap 数据库，访问数据库的用户名和密码分别为“root”和“admin”。

① 新建工程 hello5，展开工程结构图，将连接 MySQL 所需要的驱动添加到 WebContent/WEB - INF/lib 目录下，添加成功后如图 4 - 2 所示。

图 4 - 2　hello5 工程结构图

② 在 WebContent 目录下添加 conn.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>MySQL 数据库的链接</title>
</head>
<body>
<%
    String dbDriver="com.mysql.jdbc.Driver";//第 1 行代码
    String dbUrl="jdbc:mysql://localhost:3306/cap";//第 2 行代码
    String dbUser="root";//第 3 行代码
    String dbPass="admin";//第 4 行代码
    Connection conn=null;//第 5 行代码
```

```
    Class.forName(dbDriver);//加载驱动   //第 6 行代码
    conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);//获得链接   //第 7 行代码
    if(conn! = null)
        out.print("数据库链接成功");
    else
        out.print("数据库链接失败");
%>
</body>
</html>
```

代码解释：第 1 行代码定义了 MySQL 数据库的驱动字符串为“com. mysql. jdbc. Driver”；第 2 行代码定义连接 MySQL 数据库的 URL，其中指定了使用的协议为 jdbc，需要连接的数据库为 mysql，连接的服务器为 localhost(本机)，数据库为 cap；第 3、4、5 行代码定义了连接 mysql 数据使用的用户名和密码，第 5 行代码定义了 Connection 对象 conn，初始化为 null；第 6、7 行代码首先根据驱动字符串加载驱动，然后通过 dbUrl 连接数据库，剩下的代码判断 conn 对象，如果 conn 不为空，说明连接数据库成功，否则失败。

③ 连接成功后的运行结果如图 4 - 3 所示。

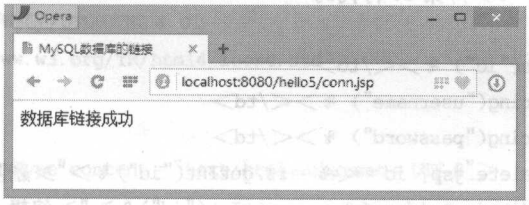


图 4 - 3　使用 JDBC 连接 MySQL 数据库成功

4.4　脚本方式实现数据库增删改查(CRUD)

本案例主要通过用脚本的方式来实现对数据库某一张表的操作，包括添加、删除、修改、查询。

数据库连接在 do_query. jsp 页面中实现，通过执行 sql 语句“select* from admin order by id”返回 user 表中的信息到 ResultSet，并显示到页面上。使用超链接的形式将需要修改、删除的记录值分别传递到 do_edit. jsp 和 do_delete. jsp 页面。

① 继续在工程 hello5 中采用 JSP 脚本方式实现对 admin 表的增删改查。首先实现查询表中的所有数据，在 WebContent 下创建 do_query. jsp，编辑代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<%@page import = "java.sql. * " %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>查询所有的记录</title>
</head>
<body>
<%
    //链接 MySQL 的驱动
    String dbDriver = "com.mysql.jdbc.Driver";//第 1 行代码
    String dbUrl = "jdbc:mysql://localhost:3306/cap";//第 2 行代码
    String dbUser = "root";//第 3 行代码
    String dbPass = "admin";//第 4 行代码
    Class.forName(dbDriver);//加载驱动//第 5 行代码
    Connection conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);//第 6 行代码
    String sql = "select * from admin order by id";//第 7 行代码
    Statement stmt = conn.createStatement();//第 8 行代码
    ResultSet rs = stmt.executeQuery(sql);//第 9 行代码
    %>
    <table border = "1">
    <% while(rs.next()){ %>//第 10 行代码
    <tr>
    <td><% = rs.getInt("id") %></td>
    <td><% = rs.getString("username") %></td>
    <td><% = rs.getString("password") %></td>
    <td><ahref = "do_delete.jsp? id = <% = rs.getInt("id") %>">删除</a></td>
    <td><ahref = "do_edit.jsp? id = <% = rs.getInt("id") %>">编辑</a></td>
    </tr>
    <% } %>
</table>
<ahref = "addAdmin.jsp">添加用户</a>
</body>
</html>
```

代码解释:第 1～6 行代码的解释和上一个案例相同。第 7 行代码定义要执行的 SQL 语句。第 8 行代码通过 conn 对象创建 Statement 对象 stmt。第 9 行代码调用 executeQuery 方法进行数据库查询,查询的结果集存放在 ResultSet 对象 rs 中。第 10 行代码判断结果集是否为空,如果不为空,进行循环画表格中的列,并通过 rs 对象的 get 方法取出相对于字段的数据,通过表达式显示到页面。

② 查询 admin 表中所有记录的结果如图 4-4 所示。

③ 实现删除功能。新建 do_delete.jsp 页面,当删除主键为 7 的记录后,页面会重现跳转到执行查询表中所有记录的页面,编辑后代码如下。

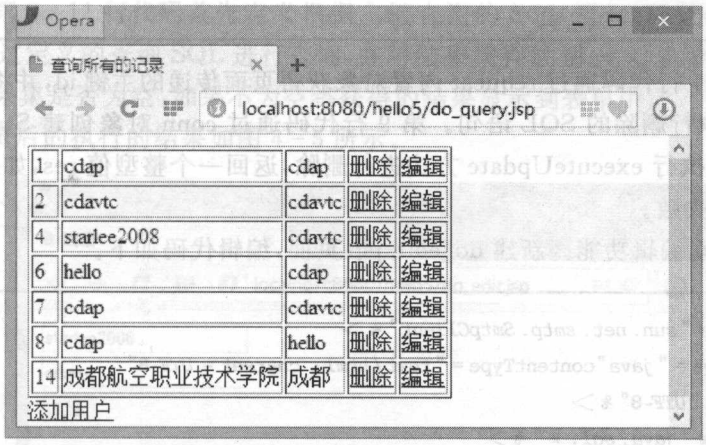


图 4-4　do_query.sjp 的运行结果

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>删除</title>
</head>
<body>
<%
    int id = Integer.parseInt(request.getParameter("id"));//第 1 行代码
    String dbDriver = "com.mysql.jdbc.Driver";//链接 MySQL 的驱动
    String dbUrl = "jdbc:mysql://localhost:3306/cap";
    String dbUser = "root";
    String dbPass = "admin";
    Class.forName(dbDriver);//加载驱动
    Connection conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);
    String sql = "delete from admin where id=" + id;//第 8 行代码
    //System.out.println("---" + sql + "-----");
    Statement stmt = conn.createStatement();//第 9 行代码
    int res = stmt.executeUpdate(sql);//第 10 行代码
    if (res < 0)
        out.print("删除失败");
    else
        response.sendRedirect("do_query.jsp");
%>
</body>
```

```
</html>
```

代码解释：第 1 行代码通过 request 内置对象获得页面传递的主键 id，并将其转化为整形。第 8 行代码定义执行删除的 SQL 语句。第 9 行代码通过 conn 对象创建 Statement 对象 stmt。第 10 行代码执行 executeUpdate 方法进行删除，返回一个整型值 res，如果 res 大于 0，说明删除成功，反之失败。

④ 第三步实现编辑功能。新建 do_edit.jsp 页面，编辑代码如下。

```
<%@page import = "sun.net.smtp.SmtpClient" %>
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<%@page import = "java.sql.* " %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>编辑</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));//第 1 行代码
        String dbDriver = "com.mysql.jdbc.Driver";//链接 MySQL 的驱动
        String dbUrl = "jdbc:mysql://localhost:3306/cap";
        String dbUser = "root";
        String dbPass = "admin";
        Class.forName(dbDriver);//加载驱动
        Connection conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);
        String sql = "select * from admin where id = " + id;//第 8 行代码
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
        if (rs.next()) {//第 11 行代码
    %>
    <form action = "do_update.jsp? id = <% = rs.getInt("id") %>"method = "post">
        <input type = "text"name = "username"value = <% = rs.getString("username") %>><br>
        <input type = "text"name = "password"value = <% = rs.getString("password") %>><br>
        <input type = "submit"value = "更新">
    </form>
    <%
        }
    %>
</body>
</html>
```

代码解释：第 8～11 行代码首先定义根据主键查询的 SQL 语句，接着创建 Statement 对象 stmt，然后通过定义的查询 SQL 进行查询，并将结果集存放到 rs 对象中。第 11 行后的代码，首先判断结果集是否为空，如果不为空将获得的结果显示到表单中。

⑤ 单击编辑后的执行的结果如图 4－5 所示。

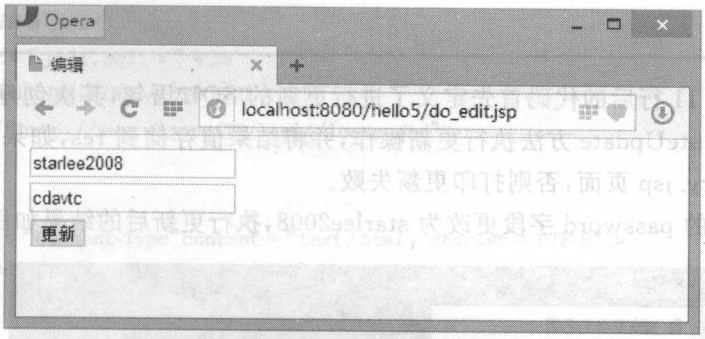


图 4－5　do_edit.jsp 运行的结果

⑥ 实现更新功能。新建 do_update.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>更新</title>
</head>
<body>
<%
    request.setCharacterEncoding("utf-8");//第 1 行代码
    int id=Integer.parseInt(request.getParameter("id"));
    String username=new String(request.getParameter("username"));
    String password=new String(request.getParameter("password"));
    String dbDriver="com.mysql.jdbc.Driver";//链接 MySQL 的驱动
    String dbUrl="jdbc:mysql://localhost:3306/cap";
    String dbUser="root";
    String dbPass="admin";
    Class.forName(dbDriver);//加载驱动
    Connection conn=DriverManager.getConnection(dbUrl, dbUser, dbPass);
    //下 1 行为第 11 行代码
    String sql="update admin set username='"+username+"',password='"+password+"' where id="+id;
    Statement stmt=conn.createStatement();
int res=stmt.executeUpdate(sql);
if(res>0)
```

```
    response.sendRedirect("do_query.jsp");
else
    out.print("更新失败");
%>
</body>
</html>
```

代码解释：第 11 行后的代码首先定义了进行更新的 SQL 语句，其次创建 Statement 对象 stmt，再调用 executeUpdate 方法执行更新操作，并将结果值存储到 res，如果 res 的值大于 0，则跳转到 do_query.jsp 页面，否则打印更新失败。

⑦ 主键为 8 的 password 字段更改为 starlee2008，执行更新后的结果如图 4-6 所示。

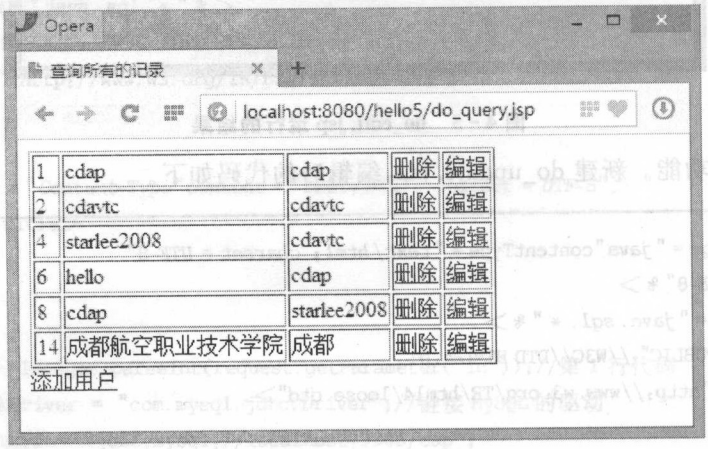


图 4-6　do_update.jsp 更新主键为 8 运行的结果

⑧ 实现向数据库中添加记录。新建 addAdmin.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>添加用户</title>
</head>
<body>
    <form action="do_insert.jsp"method="post">
    <input type="text"name="username"><br>
    <input type="text"name="password"><br>
    <input type="submit"value="添加">
</form>
</body>
```

```
</html>
```

⑨ 实现添加功能，新建 do_insert.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>添加用户</title>
</head>
<body>
<%
    String username=request.getParameter("username");//第 1 行代码
    String password=request.getParameter("password");//第 2 行代码
    String dbDriver="com.mysql.jdbc.Driver";//链接 MySQL 的驱动
    String dbUrl="jdbc:mysql://localhost:3306/cap";
    String dbUser="root";
    String dbPass="admin";
    Class.forName(dbDriver);//加载驱动
    Connection conn=DriverManager.getConnection(dbUrl, dbUser, dbPass);
    //下 1 行为第 9 行代码
    String sql="insert into admin(username,password) values('"+username+"','"+password+"')";
    Statement stmt=conn.createStatement();
    int res=stmt.executeUpdate(sql);
    if(res>0)
        response.sendRedirect("do_query.jsp");
    else
        out.print("添加失败");
%>
</body>
</html>
```

代码解释：第 1～2 行代码首先获得表单传递的用户名和密码参数；第 9 行后的代码首先定义进行添加的 SQL 语句，其次通过 conn 对象创建 Statement 对象 rs，最后进行更新操作，如果更新成功则跳转到 do_query.jsp 页面，否则显示添加失败。

⑩ 如图 4－7 所示，输入一条新记录信息，单击添加后运行结果如图 4－8 所示。

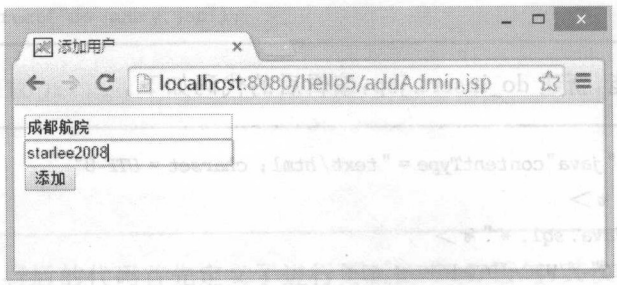


图 4-7　添加新记录

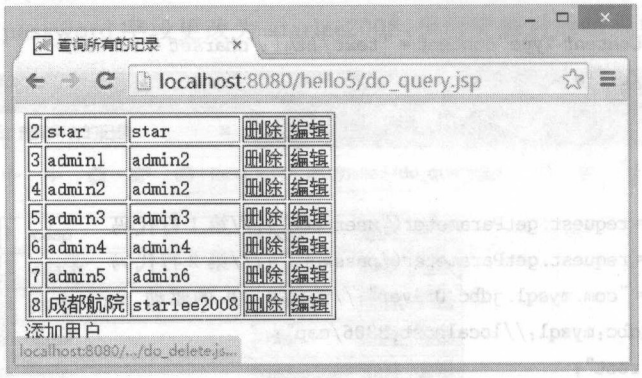


图 4-8　运行结果图

4.5　封装类实现增删改查

在 4.4 节讲的数据库操作过程中，需要在 JSP 页面完成执行之前关闭与数据库的连接。如果不关闭连接，在等待页实例被垃圾回收处理期间可能会导致数据库连接超过连接限制。且将连接代码放在页面中，不利于扩展，不利于连接的共享，性能效率低下；将连接字符串、SQL 语句暴露在页面中，不利于数据库安全；连接代码需要重复书写，不利于代码重用等诸多不利因素，影响程序的复用性和软件的灵活性。因此将数据库访问代码用类封装起来，可以很好的避免以上问题。

① 新建 Dynamic Web Project 工程 hello6，将 MySQL 的数据库驱动文件复制到 WebContent/WEB-INF/lib 目录下。

② 在 Java Resources/src 目录 cap.db 下新建 DBCon.java 类，用于封装数据库连接，如图 4-9所示。

编辑后的代码如下。

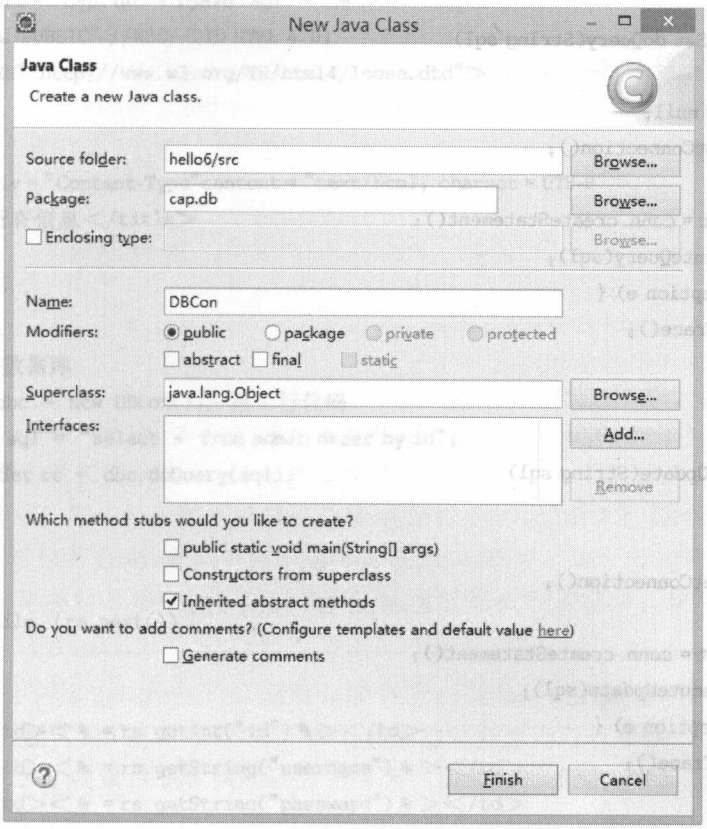


图 4-9　类创建向导

```
package cap.db;
import java.sql.*;
public class DBCon {
    private String dbDriver = "com.mysql.jdbc.Driver";//第 4 行代码
    private String dbUrl = "
    jdbc:mysql://localhost:3306/cap? useUnicode = true&characterEncoding = utf-8";
    private String dbUser = "root";
    private String dbPass = "admin";
    private Connection conn = null;      //第 8 行代码
    public Connection getConnection(){
    try {
    Class.forName(dbDriver);
    conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);
    } catch (ClassNotFoundException e) {
    e.printStackTrace();
    } catch (SQLException e) {
    e.printStackTrace();
    }
    returnconn;
```

```
    }
    public ResultSet doQuery(String sql)
    {
    ResultSet rs = null;
    conn = this.getConnection();
    try {
    Statement stmt = conn.createStatement();
    rs = stmt.executeQuery(sql);
    } catch (Exception e) {
    e.printStackTrace();
    }
    return rs;
    }
    public int doUpdate(String sql)
    {
    int res = 0;
    conn = this.getConnection();
    try {
    Statement stmt = conn.createStatement();
    res = stmt.executeUpdate(sql);
    } catch (Exception e) {
    e.printStackTrace();
    }
    return res;
    }
    public void close() throws SQLException
    {
    try {
    conn.close();
    } catch (Exception e) {
    e.printStackTrace();
    }
    }
}
```

代码解释：第 4～8 行代码首先定义连接 MySQL 数据库相关的变量，包括数据库驱动字符串，连接 MySQL 数据的 URL 字符串，用户名，密码，还有 Connection 对象 conn。getConnection 方法完成 MySQL 数据库驱动的加载后进行数据库的连接。doQuery 方法实现查询操作，doUpdate 方法实现更新操作，包括 insert，delete，update 等，close 方法关闭相关的对象，释放内存。

③ 实现查询表中的全部数据，创建 do_query.jsp，编辑后的代码如下。

```
<%@page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

```
<%@page import = "cap.db.*,java.sql.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>查询所有信息</title>
</head>
<body>
    <%
        //链接数据库
        DBCon dbc = new DBCon();//第 1 行代码
        String sql = "select * from admin order by id";
        ResultSet rs = dbc.doQuery(sql);
    %>
    <table>
        <%
            while (rs.next()) {
        %>
        <tr>
            <td><% = rs.getInt("id") %></td>
            <td><% = rs.getString("username") %></td>
            <td><% = rs.getString("password") %></td>
            <td><ahref = "do_delete.jsp? id = <% = rs.getInt("id") %>">删除</a></td>
            <td><ahref = "do_edit.jsp? id = <% = rs.getInt("id") %>">编辑</a></td>
        </tr>
        <%
            }
        dbc.close();
        %>
    </table>
    <ahref = "addAdmin.jsp">添加用户</a>
</body>
</html>
```

代码解释：第 1 行代码创建 DBCon 对象 dbc，通过调用 DBCon 的构造函数进行数据库的连接。余下的代码和前面类似，不再重复。

④ 实现删除功能，新建 do_delete.jsp，编辑后的代码。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<%@page import = "cap.db.*,java.sql.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>删除</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));//第 1 行代码
        DBCon dbc = new DBCon();
        String sql = "delete from admin where id = " + id;
        int res = dbc.doUpdate(sql);
        if (res > 0)//第 5 行代码
            response.sendRedirect("do_query.jsp");
        else
            out.println("删除失败");
    %>
</body>
</html>
```

代码解释:第 1 行代码通过 request 内置对象或者从 do_query.jsp 页面传递过来的参数 id 并转化成 int 类型。第 2 行代码创建数据库操作类 DBCon 的对象 dbc。第 3 行代码根据第 1 行代码获得的主键 id 创建删除 sql 语句。第 4 行代码调用 dbc 对象的 doUpdate 方法进行删除操作。第 5～8 行后的代码根据第 4 行代码的返回值进行判断,如果返回值大于 0,则说明删除成功,如果小于 0 则说明删除失败。

⑤ 实现编辑记录操作,新建 do_edit.jsp,编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8"%>
<%@page import = "cap.db.*,java.sql.*"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>编辑</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));//第 1 行代码
        DBCon dbc = new DBCon();
        String sql = "select * from admin where id = " + id;//第 3 行代码
        ResultSet rs = dbc.doQuery(sql);
        if (rs.next()) {//第 5 行代码
    %>
```

```
    <form action = "do_update.jsp" method = "post">
    <input type = "hidden"name = "id"value = "<% = rs.getInt("id") %>">
    <input type = "text"name = "username"value = "<% = rs.getString("username") %>">
    <input type = "text" name = "password"value = "<% = rs.getString("password") %>">
    <input type = "submit"value = "更新">
    </form>
<%
        }
        dbc.close();
    %>
</body>
</html>
```

代码解释:第 1 行代码根据 request 对象或者 do_query.jsp 页面传递过来的参数并转换为整型存储到 id 中。第 3 行代码是根据主键 id 创建查询的 sql 语句。第 4 行代码调用 dbc 对象的 doQuery 方法进行查询,查询的结果集存放在 rs 对象中。第 5 行代码判断结果集 rs 是否为空,余下的代码将结果集中的数据通过表达式显示到页面表单中。最后一行代码关闭数据库链接。

⑥ 实现更新记录操作,新建 do_update.jsp,编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<%@page import = "cap.db.*,java.sql.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>更新</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));//第 1 行代码
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        DBCon dbc = new DBCon();
        String sql = "update admin set username='" + username
                + "',password='" + password + "' where id = " + id;
        int res = dbc.doUpdate(sql);//第 6 行代码
        if (res > 0)
            response.sendRedirect("do_query.jsp");
        else
            out.println("更新失败");
    %>
```

```
</body>
</html>
```

代码解释:第 1 行代码通过 request 内置对象获得 do_query.jsp 传递过来的 id 参数,并将其转化为整形。第 2～3 行代码通过 request 内置对象获得传递过来的用户名和密码。第 4 行代码创建数据库连接 DBCon 的对象 dbc。第 5～6 行代码通过获得的用户名、密码和主键创建更新 SQL 语句。第 6 行代码调用 dbc 对象的 doUpdate 方法实现更新,并将返回值存放在 res 中,如果返回值 0,则说明更新成功并跳转到 do_query.jsp 页面,否则,更新失败。

⑦ 实现添加功能,新建 addAdmin.jsp 和 do_insert.jsp 页面,addAdmin.jsp 的页面和4.4 节的代码相同,请参考 4.4 节或者本书提供的源码。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="cap.db.*,java.sql.*"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>添加用户</title>
</head>
<body>
    <%
        request.setCharacterEncoding("utf-8");//第 1 行代码
        String username = new String(request.getParameter("username"));
        String password = new String(request.getParameter("password"));
        DBCon dbc = new DBCon();
        String sql = "insert into admin(username,password) values('"
                + username + "','" + password + "')";
        int res = dbc.doUpdate(sql);//第 6 行代码

        if (res > 0)
            response.sendRedirect("do_query1.jsp");
        else
            out.println("添加失败");
    %>
</body>
</html>
```

代码解释:第 1 行代码设置请求页面的字符编码为 UTF-8,目的是为了防止提交中文的时候出现乱码。第 2～3 行代码获得用户名和密码,并通过 String 的构造方法将其转为UTF-8 的编码。第 4 行代码创建数据库连接对象 dbc。第 5～6 行代码根据获得的用户名和密码创建添加的 sql 语句。第 6 行代码调用 dbc 对象的 doUpdate 方法实现添加,并将返回值存放在

res 值中，接着判断返回值 res 如果大于 0，则跳转到 do_query.jsp 页面，否则在页面上提示添加失败。

⑧ 本案例中实现的效果同 4.4 节，即每一步的操作运行结果图对应参照 4.4 节。

4.6　脚本实现数据库分页显示

当查询结果超过一屏的显示范围，则需要使用分页的形式显示，本案例将展示使用脚本来实现数据库分页显示。

① 新建 Dynamic Web Project 工程 hello7，将 MySQL 的数据库驱动文件复制到 WebContent/WEB-INF/lib 目录下。将 hello6 中的 db 包下的 DBCon 类复制到 hello7 中。

② 新建 pager.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.sql.*,cap.db.*"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>分页显示</title>
</head>
<body>
    <%
        //总的页面数
        int pageCount = 0;//第 1 行代码
        //每页显示的记录数
        int pageSize = 5;
        //当前页面
        int currentP = 1;
        //表中的记录数
        int totalCount = 0;
        //数据库链接
        DBCon dbc = new DBCon();//第 5 行代码
        //表中总记录数
        String sql = "select count(*) as t from admin";
        ResultSet rs = dbc.doQuery(sql);
        if (rs.next())
            totalCount = rs.getInt("t");//第 9 行代码
        /*
        根据总的记录数来计算页面数
         */
        if (totalCount / pageSize == 0)//第 10 行代码
```

```
            pageCount = (totalCount / pageSize);
        else
            pageCount = (totalCount / pageSize) + 1;//第 13 行代码
        //获得分页条上的当前页码
        String pStr = request.getParameter("p");//第 14 行代码
        if (pStr == null)
            pStr = "1";
        currentP = Integer.parseInt(pStr);
        //如果当前页大于总的页面数,当前页面赋值为总的页面数
        if (currentP > pageCount)
            currentP = pageCount;
        //如果当前页小于 0 重置为第一页
        if (currentP < 0)
            currentP = 1;//第 21 行代码
        //分页查询,mysql 的分页查询关键字是 limit,注意 limit 后面有空格
        sql = "select * from admin limit " + (currentP - 1) * pageSize
                + "," + pageSize;//第 22 行代码
        rs = dbc.doQuery(sql);//第 23 行代码
    %>
    <table>
        <%
            while (rs.next()) {
        %>
        <tr>
            <td><% = rs.getInt("id") %></td>
            <td><% = rs.getString("username") %></td>
            <td><% = rs.getString("password") %></td>
        </tr>
        <%
            }
        %>
    </table>
    <%
        //分页条
        if (currentP > 1) {
    %>
    <ahref = "pager.jsp? p = 1">第一页</a>
    <ahref = "pager.jsp? p = <% = currentP - 1 %>">上一页</a>
    <%
        }
    %>
    <%
        if (currentP <= pageCount) {
    %>
    <ahref = "pager.jsp? p = <% = currentP + 1 %>">下一页</a>
```

```
    <ahref = "pager.jsp? p = < % = pageCount % >">最后一页</a>
    < % } % >
</body>
</html>
```

代码解释：第 1～4 行代码，其中变量 pageCount 存放页面的总数目，变量 pageSize 存放每页要显示的记录数，变量 currentP 表示当前页面，变量 totalCount 表示存放查询表中的总记录数。第 5～9 行代码查询表中总的记录数，并存储在变量 totalCount 中。

第 10～13 行代码根据查询的总记录数来计算页面数，如果 totalCount / pageSize == 0，则页面数为 totalCount / pageSize，否则为(totalCount / pageSize)+1。例如查询的总记录数 totalCount 为 21，每页显示的记录 pageSize 为 10，则根据上面的公式计算出总页数为 2+1=3 页，如果总的记录数 totalCount 为 20，同样每页显示的记录数为 10，则总的页面数 pageCount 的值为 2。

典型的分页算法如下代码所示：

```
if (totalCount / pageSize = = 0)
    pageCount = (totalCount / pageSize);
else
    pageCount = (totalCount / pageSize) + 1;
```

第 14～21 行代码获得传递过来的当前页面值，如果第一次运行 page.jsp 时候，获得 pStr 的值为 null，所以将 pStr 的值赋为 1，并将其转为整型赋给 currentP，在分页的时候接着判断如果当前页面数小于 0，则值赋为 1，如果当前页面数大于最大页面数，则复制为最大页面数。第 22～23 行代码根据当前页 currentP 和每页显示的记录 pageSize 创建分页的 sql 语句，注意在 MySQL 中分页的关键字为 limit，不同的数据库分页的关键字也不同。第 24～28 行代码根据分页 sql 语句进行查询，并通过表达式显示到页面，余下的代码在页面实现分页条。

③ 结果显示，如图 4－10 所示。

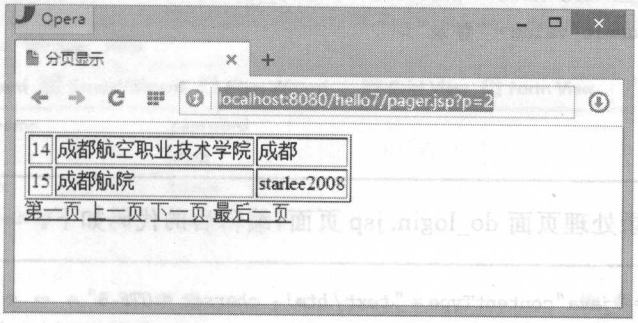


图 4－10　分页显示结果

注意：在第本章中采用的字符串拼接生成的 SQL 语句，会遭遇到 SQL 注入攻击的可能性，所以建议采用本书后面介绍的技术，比如预编译、JavaBean 或者 MVC 技术实现。

4.7 预编译进行数据库的增删改查

4.7.1 SQL 注入攻击概念

SQL 注入攻击是黑客对数据库进行攻击的常用手段之一。随着 B/S 模式应用开发的发展,使用 B/S 模式编写应用程序的程序员也越来越多。但是由于程序员的水平及经验也参差不齐,相当大一部分程序员在编写代码时,没有对用户输入数据的合法性进行判断,使应用程序存在安全隐患。用户可以提交一段数据库查询代码,根据程序返回的结果,获得某些他想得知的数据,这就是所谓的 SQL Injection,即 SQL 注入。

4.7.2 SQL 注入攻击案例

本案例中主要通过伪造一个 SQL 语句,实现对数据库的 SQL 注入攻击。

① 在工程 hello7/WebContent 下添加 login.jsp 页面,编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>SQL 注入攻击案例</title>
</head>
<body>
    <form action="do_login.jsp"method="post">
    <input type="text"name="username"><br>
    <input type="password"name="password"><br>
    <input type="submit"value="登录">
    </form>
</body>
</html>
```

② 继续添加登录处理页面 do_login.jsp 页面,编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="java.sql.*,cap.db.*"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>SQL 注入攻击演示</title>
</head>
<body>
    <%
        DBCon dbc = new DBCon();
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String sql = "select * from admin where username='" + username
                + "' and password='" + password + "'";
        ResultSet rs = dbc.doQuery(sql);
    %>
    <table>
        <%
            while (rs.next()) {
        %>
        <tr>
            <td><% = rs.getInt("id") %></td>
            <td><% = rs.getString("username") %></td>
            <td><% = rs.getString("password") %></td>
        </tr>
        <%
            }
        %>
    </table>
</body>
</html>
```

③ 在数据库 cap 中存在如图 4-11 所示的用户数据。

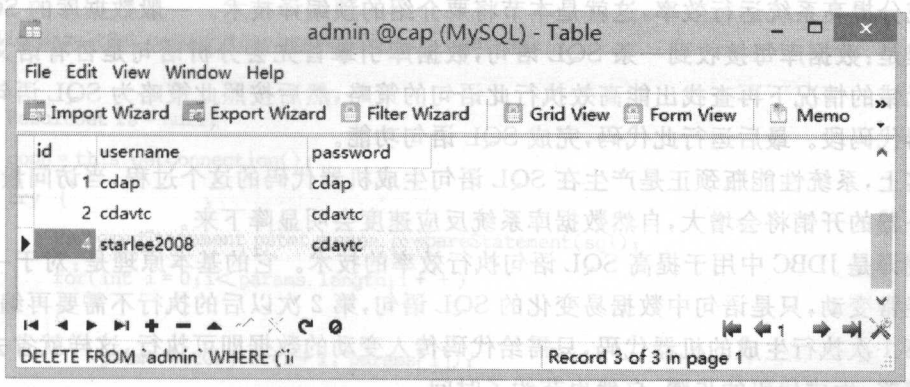


图 4-11　admin 表中的用户记录

当用户在登录页面输入如图 4-12 所示的精心构造的字符串。

do_login.jsp 页面会显示如图 4-13 的结果。

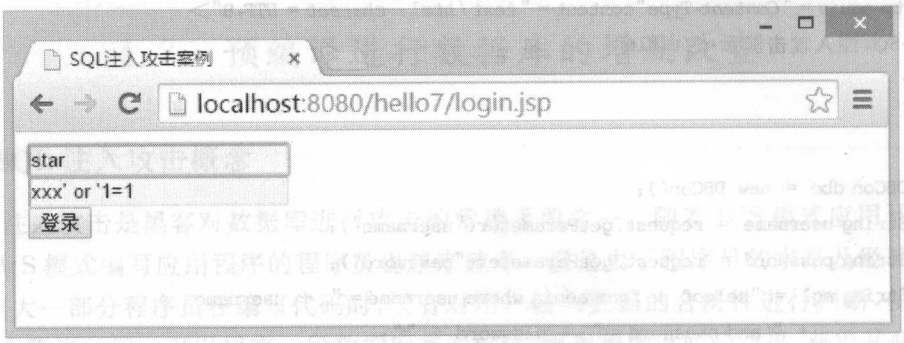


图 4-12　用户的输入

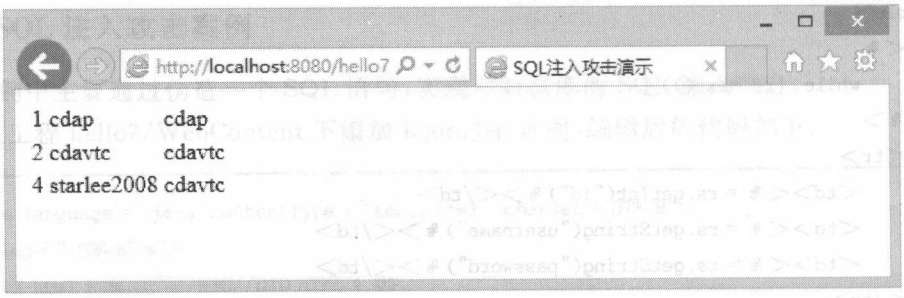


图 4-13　登录的结果

从上面可以看出，采用精心构造的字符串，即使输入错误的用户名和密码，一样可以获得数据库表中的信息，所以说应该采用更为安全技术手段来防止 SQL 注入攻击，下面介绍预编译技术，可以有效的避免 SQL 注入攻击。

4.7.3　预编译概念

数据库连接操作是影响 JSP 系统性能的一个重要因素，通过优化数据库操作语句的执行过程可充分提高系统运行效率，这就是本节将要介绍的预编译技术。一般数据库的 SQL 语句执行过程是：数据库每接收到一条 SQL 语句，数据库引擎首先会分析语句是否有语义或语法错误。无错的情况下再查找出能高效执行此语句的策略，然后按照此策略为 SQL 语句生成正确的机器代码段。最后运行此代码，完成 SQL 语句功能。

事实上，系统性能瓶颈正是产生在 SQL 语句生成机器代码的这个过程，当访问量增大，数据库服务器的开销将会增大，自然数据库系统反应速度会明显降下来。

预编译是 JDBC 中用于提高 SQL 语句执行效率的技术。它的基本原理是：对于一条语法结构不经常变动，只是语句中数据易变化的 SQL 语句，第 2 次以后的执行不需要再编译，而直接使用第 1 次执行生成的机器代码，只需给代码传入变动的数据即可执行，这样就省去每次都要查找策略、生成代码的步骤，自然也节约了时间。

预编译的优点还包括非动态创建字符串，允许 Java 虚拟机(JavaVirtual Machine，JVM)和驱动/数据库缓存语句和字符串并提高性能、提供数据库无关性、减少 SQL 语句的数据库依赖性等。

4.7.4　预编译使用案例

① 新建 Dynamic Web Project 工程 hello8，将 MySQL 的数据库驱动文件复制到 WebContent/WEB-INF/lib 目录下。

② 在 db 包里新建 DBCon.java，编辑后的代码如下。

```
package cap.db;
import java.sql.*;
import java.util.*;
public class DBCon {
    private static String dbDriver = "com.mysql.jdbc.Driver";
    private String dbUrl = "jdbc:mysql://localhost:3306/cap? useUnicode=true&characterEncoding=utf-8";
    private String dbUser = "root";
    private String dbPass = "admin";
    private Connection conn = null;
    public Connection getConnection()
    {
        try {
            Class.forName(dbDriver);
            conn = DriverManager.getConnection(dbUrl,dbUser,dbPass);
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        returnconn;
    }
    public ResultSet doQuery(String sql,Object[] params)
    {
        ResultSet rs = null;
        conn = this.getConnection();
        try {
            PreparedStatement pstmt = conn.prepareStatement(sql);
            for(int i = 0;i<params.length;i++)
            {
                pstmt.setObject(i+1, params[i]);
            }
            rs = pstmt.executeQuery();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        return rs;
    }
    public int doUpdate(String sql,Object[] params)
    {
        int res = 0;
        conn = this.getConnection();
        try {
            PreparedStatement pstmt = conn.prepareStatement(sql);
            for(int i = 0;i<params.length;i++)
            {
                pstmt.setObject(i + 1, params[i]);
            }
            res = pstmt.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return res;
    }
    public List<Object> doQueryList(String sql,Object[] params)
    {
        List<Object> list = new ArrayList<Object>();
        ResultSet rs = this.doQuery(sql, params);
        try {
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnLength = rsmd.getColumnCount();//获得表结构的字段个数

            while(rs.next())
            {
                Map<String,Object> map = new HashMap<String,Object>();
                for(int i = 1;i<=columnLength;i++)
                {
                    map.put(rsmd.getColumnLabel(i), rs.getObject(i));
                }
                list.add(map);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return list;
    }
```

```
    public void close()
    {
        try {
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

代码解释：getConnection 方法首先加载 MySQL 的链接驱动，然后进行数据库连接并返回 Connection 对象。doQuery 方法，需要传递要执行的 SQL 语句以及执行查询时的参数数组，执行完成后返回 ResultSet 对象。doUpdate 方法进行更新操作，同样需要传递执行的 SQL 语句以及执行更新操作需要的参数数组，返回受影响的行数 res。doQueryList 方法和 doQuery 类似，区别是 doQueryList 方法将取得的结果存放在 Map 对象中，然后再存放在 List 数组中。

③ 实现查询所有的记录的操作，新建 do_query.jsp，编辑后的代码如下。

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="cap.db.*,java.util.*"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>查询所有的记录</title>
</head>
<body>
<%
    DBCon dbc=new DBCon();
    String sql="select * from admin";
    List list=dbc.doQueryList(sql, new Object[]{});
%>
<table border="1">
<%for(int i=0;i<list.size();i++){
    Map<String,Object> map=(Map<String,Object>)list.get(i);
%>
<tr>
    <td><%=map.get("id")%></td>
    <td><%=map.get("username")%></td>
    <td><%=map.get("password")%></td>
    <td><ahref="do_delete.jsp?id=<%=map.get("id")%>">删除</a></td>
```

```
    <td><ahref = "do_edit.jsp? id = <% = map.get("id") %>">编辑</a></td>
    </tr>
    <%}
    dbc.close();
    %>
    </table>
    <ahref = "adduser.jsp">添加用户</a>
</body>
</html>
```

代码解释:第 1 个<%%>代码块,首先创建数据库链接类 DBConnection 的对象 dbc,接着调用 dbc 对象的 doQueryList 方法,结果集存放在 List 类型的 list 对象。第 2 个<%%>代码块将 list 中的数据遍历取出,存放在 Map 类型中。余下的代码采用表达式将结果显示到页面上。

④ 根据主键删除指定的记录,新建 do_delete.jsp,编辑代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<%@page import = "java.sql.*,cap.db.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>删除</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));//第 1 行代码
        DBCon dbc = new DBCon();
        String sql = "delete from admin where id = ?";
        int res = dbc.doUpdate(sql, new Object[] { id });//第 4 行代码
        if (res > 0)
            response.sendRedirect("do_query.jsp");
        else
            out.println("删除失败");
    %>
</body>
</html>
```

代码解释:第 1~3 行代码和前面的解释相似。第 4 行代码调用 dbc 对象的 doUpdate 方法进行删除,注意到此方法传递的参数是数组,可以根据不同的情况传递参数。后面的代码也不再详述。

⑤ 实现根据主键编辑指定记录的操作，新建 do_edit.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="cap.db.*,java.sql.*"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN""http://www.w3.org/TR/html4/loose.
dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>编辑</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));
        DBCon dbc = new DBCon();
        String sql = "select * from admin where id=?";
        ResultSet rs = dbc.doQuery(sql, new Object[] { id });
        if (rs.next()) {
    %>
        <form action="do_update.jsp"method="post">
            <input type="hidden"name="id"value="<%=rs.getInt("id")%>">
                <input type="text"name="username"value="<%=rs.getString("username")%>">
            <input type="text"name="password"value="<%=rs.getString("password")%>">
            <input type="submit"value="更新">
        </form>
    <%
        }
        dbc.close();
    %>
</body>
</html>
```

代码解释：参看 4.5 节详述。

⑥ 根据指定的主键更新纪录，创建 do_update.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="cap.db.*,java.sql.*"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
```

```
<title>更新</title>
</head>
<body>
    <%
        request.setCharacterEncoding("utf-8");
        int id = Integer.parseInt(request.getParameter("id"));
        String username = new String(request.getParameter("username"));
        String password = new String(request.getParameter("password"));
        DBCon dbc = new DBCon();
        String sql = "update admin set username = ?,password = ? where id = ?";
        int res = dbc.doUpdate(sql,new Object[]{username,password,id});
        if (res > 0)
            response.sendRedirect("do_query.jsp");
        else
            out.println("更新失败");
    %>
</body>
</html>
```

代码解释:参看 4.5 节代码解释。

⑦ 实现向数据库中表添加记录,分别创建 addAdmin.jsp 和 do_insert.jsp,编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>添加用户</title>
</head>
<body>
    <form action = "do_insert.jsp" method = "post">
    <input type = "text"name = "username"><br>
    <input type = "text"name = "password"><br>
    <input type = "submit"value = "添加">
    </form>
    </body>
</html>
```

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<%@page import = "java.sql.*,cap.db.*" %>
```

```
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN""http://www.w3.org/TR/html4/loose.
dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
    String name = request.getParameter("username");
    String pass = request.getParameter("password");
    DBCon  dbc = new DBCon();
    String sql = "insert into admin(username,password) values(?,?)";
    int res = dbc.doUpdate(sql,new Object[]{name,pass});
    if(res>0)
        response.sendRedirect("do_query.jsp");
    else
        out.print("添加失败");
%>
</body>
</html>
```

代码解释:参看 4.5 节代码解释。

⑧ 运行 do_query.jsp,效果如图 4-14 所示。

图 4-14　使用预编译查询表中所有记录

4.8　连接池

开发基于数据库的 Web 应用系统,传统的做法是针对每一次 Web 请求,应用程序必须首先创建一个到数据库的连接(Connection)。然后,这个连接就成为应用程序和数据库之间的通道,应用程序使用它将 SQL 语句发送给数据库,数据库使用它返回执行结果。一个 Con-

nection 和一个数据库用户帐号相关联，这样可以使数据库对通过 Connection 提交来的 SQL 语句执行访问控制检查。创建一个连接大概需要 1～2 秒的时间。除了建立到数据库的网络连接之外，数据库引擎还必须对用户进行身份验证，并用各种数据结构创建一个环境以便记录事务。

频繁的数据库连接操作，不仅费时耗资源，而且一个连接使用太久也会不稳定，连接池(connection pool)技术的出现就是为了解决这些问题。连接池中保存了一些 Connection 对象，这些对象被所有 Servlet 和 JSP 页面所共享。对于每个请求都会分配给它一个连接，使用完后再收回这个连接。

连接池的基本思想是预先建立一些数据库连接放置于内存对象中以备使用，当程序中需要建立数据库连接时，只需从内存中取一个来用而不用新建。同样，使用完毕后，只需放回内存即可。而连接的建立、断开都由连接池自身来管理。同时，用户还可以通过设置连接池的参数来控制连接池中的连接数、每个连接的最大使用次数等。通过使用连接池，将大大提高程序效率。同时，还可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。综上所述，连接池技术具备以下一些优点。

1. 解决了创建连接需要时间

放入池内的连接只被创建一次，以后一直重复使用这个连接。大多数连接池的实现都允许用户在启动时指定 Connection 对象的初始化个数和最大个数。如果需要就可以创建新的 Connection 对象，直到达到最大个数。一旦达到了最大个数，连接池的客户就必须等待一个已有的 Connection 对象用完，而不能再创建新的 Connection 对象。

2. 解决了共享连接造成多线程的问题

有了连接池之后，每个请求将得到它自己的 Connection 对象，所以在某一时刻它只被一个线程使用，从而避免了潜在的多线程问题。

3. 解决了连接的资源有限

有了连接池之后，每个连接都会得到有效地利用。如果有很多请求挂起，连接就很难在连接池中休息。如果连接池允许指定 Connection 对象的最大个数，还可以根据响应时间来权衡并发出连接的个数。

4.8.1 JNDI

Java 命名和目录接口(Java Naming and Directory Interface，JNDI)是一组在 Java 应用中访问命名和目录服务的 API。命名服务将名称和对象联系起来，使得可以用名称访问对象。目录服务是一种命名服务，在这种服务里，对象不但有名称，还有属性。

JNDI 类似 JDBC 构建在抽象层上，是一组在 Java 应用中访问命名和目录的 API。在使用 DataSource 或任何其他共享的资源通过 JNDI 可用方面，JEE 定义了 JNDI 这种更为灵活的方式。通过 JNDI，连接池可用于应用程序的所有部分，即使对无权访问 Servlet 环境的组件来说也可用。因此，这应该是资源共享的首选，除非选择不支持 JNDI 的容器。

为了使用 JNDI，首先要使用＜resource－ref＞元素在 web. xml 应用程序配置描述中定义资源。

```
<resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/mysql</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

可选的<description>元素描述了资源，可以用来帮助人们的部署实现。

<res-ref-name>元素是必需的，而且必须包含一个唯一的名称，应用程序组件要使用该名称检索资源。对于数据资源，JEE 规范建议使用这里展示的命名约定，如，使用 JNDIJDBC 子环境中的一个名称。

资源的类型必须由<res-type>元素定义。它必须是资源的完全限定类名，对于数据资源来说，通常是 javax.sql.DataSource。

随后就是<res-auth>元素。它接受一个或两个值：Container 或 Application。Container 意味着当数据资源注册为 JNDI 资源时，从数据源那里获取连接时需要的数据库账户信息必须提供给容器，因此容器就可以进行身份验证。Application 则意味着应用程序每次获得连接时都提供该信息。这归结为应用程序是否会调用 getConnection()(在容器控制身份验证的情况下)或 getConnection(String username,String password)(在应用程序控制的情况下)。多数情况下，应该使用容器来进行。

应用程序组件、Servlet、自定义行为、bean 或应用程序使用的任何其他类型的类，都使用 JNDI API 来获得 DataSource 和 Connection，其实现如下代码所示。

```
import javax.naming.Context;
Import javax.naming.InitialContext;
Import javax.sql.DataSource;
Import java.sql.Connection;
...
InitialContext contex = new InitialContext();
DataSource ds = (DataSource)contex.lookup("java:comp/env/jdbc/mysql");
```

InitialContext 表示的是容器的 JNDI 资源命名服务的入口点。lookup()方法参数是用于 DataSource 的路径。第一部分，java:comp/env/是所有 JEE 资源的基础，后面跟由<res-ref-name>元素在配置描述符中声明的值。通过 JNDI 检索 DataSource 之后，应用程序会照常通过调用 getConnection()得到一个 Connection。

4.8.2　连接池使用案例

要使用连接池，首先要进行相关的配置，配置的步骤如下。

① 首先展开 Servers/Tomcat v7.0 server at localhost-config，如图 4-15 所示。其次打开 context.xml，在<Context></Context>中添加如下代码。

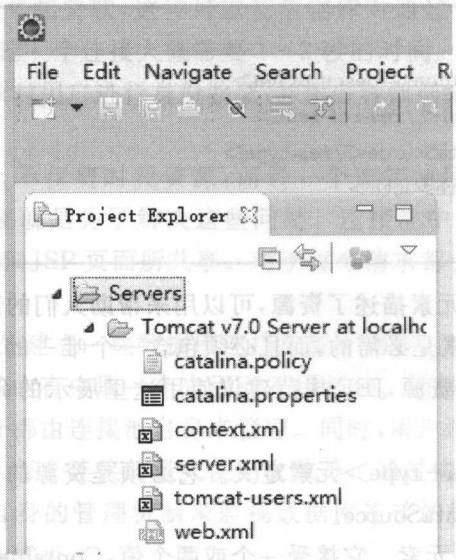


图 4-15 Servers 的结构图

```
<Resourcename = "jdbc/mysql"
    auth = "Container"
    type = "javax.sql.DataSource"
    username = "root"
    password = "admin"
    maxActive = "50"
    maxIdle = "10"
    maxWait = "5000"
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://localhost:3306/cap"
/>
```

② 在 web.xml 中<web-app>和</web-app>标签中添加下面的代码。

```
<resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/mysql</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

③ 继续在 hello8 工程中的 db.db 子包中添加 DBPool.java 类，编辑后的代码如下。

```
package cap.db;
import java.sql.*;
import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
public class DBPool {
    private String jndiName = "java:comp/env/jdbc/mysql";
    private Connection conn = null;
    public Connection getConnection() {
        try {
            InitialContext context = new InitialContext();
            DataSource ds = (DataSource) context.lookup(jndiName);
            conn = ds.getConnection();
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }
    public ResultSet doQueryRS(String sql, Object[] params){
        ResultSet rs = null;
        conn = this.getConnection();
        try {
            PreparedStatement pstmt = conn.prepareStatement(sql);
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
            rs = pstmt.executeQuery();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
    public List<Object> doQueryList(String sql, Object[] params)
            throws SQLException {
        ResultSet rs = this.doQueryRS(sql, params);
        List<Object> lists = new ArrayList<Object>();
        ResultSetMetaData rsmd = rs.getMetaData();
        int coloumn = rsmd.getColumnCount();
        while (rs.next()) {
            Map<String, Object> map = new HashMap<String, Object>();
            for (int i = 1; i <= coloumn; i++) {
                map.put(rsmd.getColumnLabel(i), rs.getObject(i));
            }
```

```
            lists.add(map);
        }
        return lists;
    }
    // 实现向数据库添加记录
    public int doUpdate(String sql, Object[] params){
        int res = 0;
        conn = this.getConnection();
        try {
            PreparedStatement pstmt = conn.prepareStatement(sql);
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
        res = pstmt.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return res;
    }
    public void close(){
        try {
            if(conn! = null)
                conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

代码解释:在 getConnction 方法中,首先初始化 InitialContext 的对象 context,然后 context 根据 jndiName 查找数据源,并返回数据源对象 ds。

④ 在工程 hello8 的 WebContent 中创建连接数据源的页面 pool_query.jsp,编辑页面的代码如下所示。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<%@page import = "cap.db.*,java.util.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>Tomcat 连接池的使用</title>
</head>
<body>
```

```
<%
String sql = "select * from admin order by id";//第 1 行代码
DBPool dbc = new DBPool();//第 2 行代码
List<Object> lists = dbc.doQueryList(sql, new Object[]{});//第 3 行代码
%>
    <table border = "1">
    <% for(int i = 0;i<lists.size();i++){
        Map<String,Object> map = (Map<String,Object>)lists.get(i);
    %>
    <tr>
    <td><% = map.get("id") %></td>
    <td><% = map.get("username") %></td>
    <td><% = map.get("password") %></td>
    <td><ahref = "do_delete.jsp? id = <% = map.get("id") %>">Delete</a></td>
    <td><ahref = "do_edit.jsp? id = <% = map.get("id") %>">Edit</a></td>
    </tr>
    <%} %>
</table>
<ahref = "addadmin.jsp">Add</a>
</body>
</html>
```

代码解释：第 1 行代码创建查询的 SQL 语句。第 2 行代码创建 DBPool 对象 dbc。第 3 行代码调用 dbc 对象的 doQueryList 方法进行查询，结果返回到类型为 List 的对象 lists 中。剩下的代码在前面已经详述，运行的结果如图 4-16 所示。

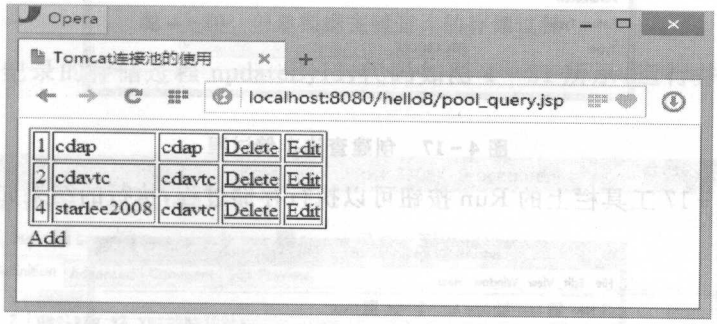


图 4-16　调用连接池查询结果图

4.9　JSP 调用存储过程

4.9.1　存储过程的概念

存储过程(Stored Procedure)是在大型数据库系统中，一组为了完成特定功能的 SQL 语句和流程控制语句的集合，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数(如果是带参数的存储过程)来执行它。存储过程具有以下的优点：

① 允许标准组件式编程：将常用或复杂的操作用一组 SQL 语句编写实现并用指定名字存储起来，后期如有请求的服务与这些定义好的存储过程功能相同的，就可调用执行。

② 能够提高执行效率：存储过程只在建立之初编译，以后的执行无需重新编译，这种已经编译好的过程可极大改善 SQL 语句性能。

③ 安全性高：存储过程可设置指定用户使用权限，可以作为一种安全机制来充分利用。

④ 减少网络通信量：客户端通过一条调用存储过程的语句，就可以完成需大量 SQL 语句才能完成的任务，减少了客户端和服务器之间大量的请求/回答包。

4.9.2 存储过程使用案例

不同的数据库在存储过程的使用上有所区别，下面以 MySQL 为例来讲解存储过程的创建以及在 JSP 中调用这个存储过程的方法。

本节主要讲述 JSP 如何调用 MySQL 的存储过程实现增删改查，首先讲述如何在 MySQL 的管理工具 Navicat 中创建存储过程。

① 启动 Navicat for MySQL，打开创建好的数据库 cap，选择 Stored Procedures/New StoredProc，创建 findAll 存储过程，如图 4-17 所示。

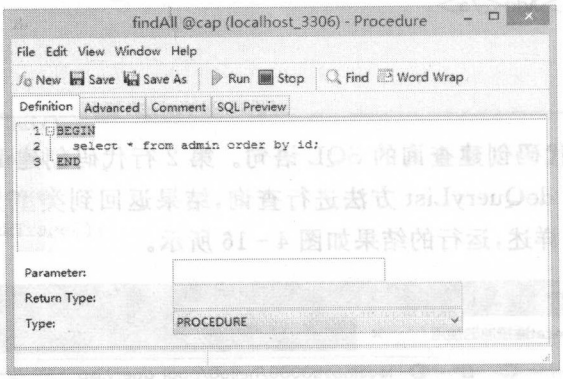


图 4-17 创建查询存储过程

② 单击图 4-17 工具栏上的 Run 按钮可以执行存储过程，执行的结果如图 4-18 所示。

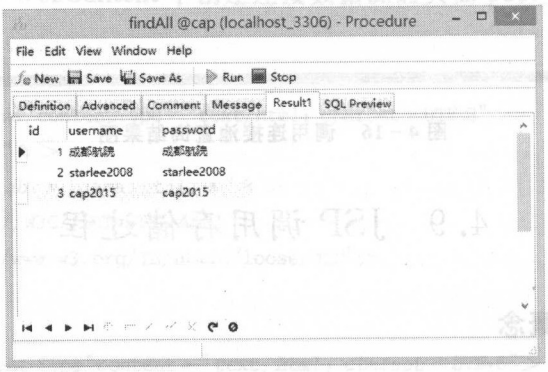


图 4-18 运行查询存储过程的结果

③ 创建删除纪录的存储过程保存为 delById，编写的代码如图 4-19 所示。

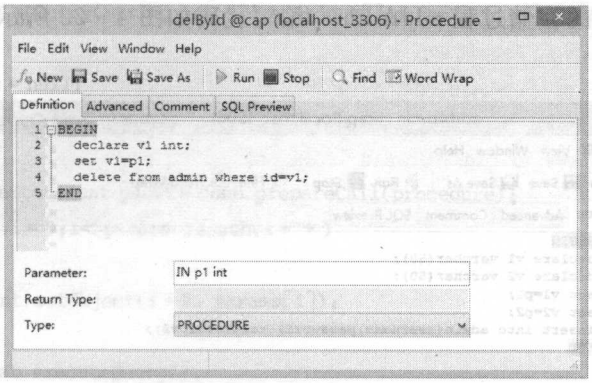


图 4－19　创建根据主键删除的存储过程

④ 创建根据主键 id 为参数查询数据的存储过程并保存为 findById，编写的代码如图 4－20 所示。

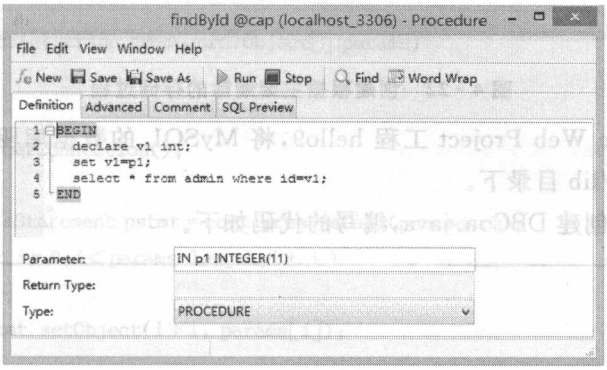


图 4－20　创建根据主键查询的存储过程

⑤ 创建更新纪录的存储过程 updateById，代码如图 4－21 所示，运行并用 findAll 查看结果。

图 4－21　创建更新的存储过程

⑥ 创建插入纪录的存储过程 addAdmin，编写代码如图 4 - 22 所示，运行并用 findAll 查看结果。

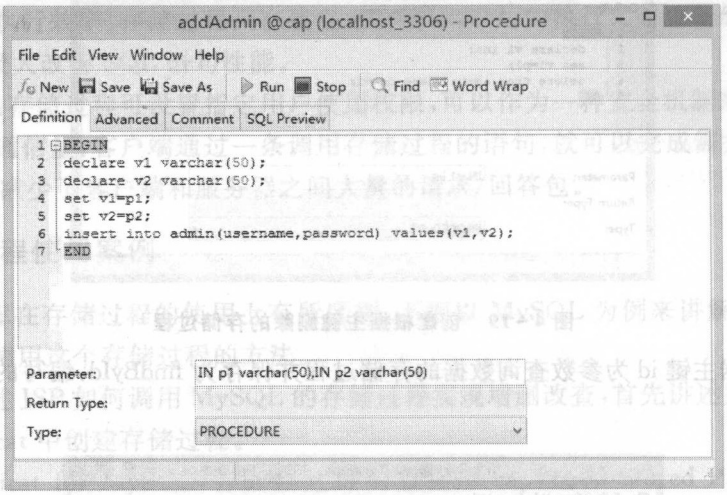


图 4 - 22　创建根据主键删除的存储过程

⑦ 新建 Dynamic Web Project 工程 hello9，将 MySQL 的数据库驱动文件复制到 WebContent/WEB - INF/lib 目录下。

⑧ 在 db 包下面创建 DBCon. java，编写的代码如下。

```
package cap.db;
import java.sql.*;
public class DBCon {
    private static String dbDriver = "com.mysql.jdbc.Driver";
    private String dbUrl = "jdbc:mysql://localhost:3306/cap? useUnicode = true&characterEncoding =
    utf-8";
    private String dbUser = "root";
    private String dbPass = "admin";
    private Connection conn = null;

    public Connection getConnection()
    {
        try {
            Class.forName(dbDriver);
            conn = DriverManager.getConnection(dbUrl,dbUser,dbPass);
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        returnconn;
    }
```

```
public ResultSet doQuery(String procedure,Object[] params)
{
    ResultSet rs = null;
    conn = this.getConnection();
    try {
        CallableStatement pstmt = conn.prepareCall(procedure);
        for(int i = 0;i<params.length;i++)
        {
            pstmt.setObject(i+1, params[i]);
        }
        rs = pstmt.executeQuery();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return rs;
}
public int doUpdate(String procedure,Object[] params)
{
    int res = 0;
    conn = this.getConnection();
    try {
        CallableStatement pstmt = conn.prepareCall(procedure);
        for(int i = 0;i<params.length;i++)
        {
            pstmt.setObject(i+1, params[i]);
        }
        res = pstmt.executeUpdate();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return res;
}
publicvoid close()
{
    try {
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
```

⑨ 调用 findAll 存储过程查询表中的所有记录，新建 call_query.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
```

```
    pageEncoding = "UTF-8" % >
< % @page import = "java.sql.ResultSet" % >
< % @page import = "java.util. * ,cap.db. * " % >
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>查询所有的记录</title>
</head>
<body>
    < %
    DBCon dbc = new DBCon();
    String procedure = "{call findAll()}";
    ResultSet rs = dbc.doQuery(procedure,new Object[]{});
% >
    <table border = "1">
    < % while(rs.next()){ % >
    <tr>
    <td>< % = rs.getInt("id") % ></td>
    <td>< % = rs.getString("username") % ></td>
    <td>< % = rs.getString("password") % ></td>

    <td><ahref = "call_edit.jsp? id = < % = rs.getInt("id") % >">编辑</a></td>
    <td><ahref = "call_delete.jsp? id = < % = rs.getInt("id") % >">删除</a></td>
    </tr>
    < % } % >
    </table>
    <ahref = "addAdmin.jsp">Add</a>
</body>
</html>
```

⑩ 调用 delById 存储过程，根据主键删除指定的记录，新建 call_delete.jsp，编辑代码如下。

```
< % @page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" % >
< % @page import = "java.sql. * ,cap.db. * " % >
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>调用存储过程实现删除</title>
</head>
```

```
<body>
<%
    int id = Integer.parseInt(request.getParameter("id"));
    DBCon dbc = new DBCon();
    String procedure = "{call delById(?)}";
    int res = dbc.doUpdate(procedure,new Object[]{id});//第 4 行代码
    if(res>0)
       response.sendRedirect("call_query.jsp");
    else
       out.println("Delete error");
    %>
</body>
</html>
```

代码解释：第 4 行代码调用存储过程实现删除，delById 是前面已经定义好的删除存储过程。参数同样采用？占位，第 4 行代码传递要删除的数据的主键。

⑪ 调用 findById 存储过程，实现根据指定的主键查询，新建 call_edit.jsp，编辑后的代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<%@page import = "java.util.*,cap.db.*" %>
<%@page import = "java.sql.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN""http://www.w3.org/TR/html4/loose.
dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>编辑</title>
</head>
<body>
<%
   int id = Integer.parseInt(request.getParameter("id"));
   String procedure = "{call findById(?)}";
   DBCon dbc = new DBCon();
   ResultSet rs = dbc.doQuery(procedure, new Object[]{id});
   if(rs.next()){
%>
<form action = "call_update.jsp"method = "post">
<input type = "hidden"name = "id"value = <% = rs.getInt("id") %>>
<input type = "text"name = "username"value = <% = rs.getString("username") %>>
<input type = "text"name = "password"value = <% = rs.getString("password") %>>
<input type = "submit"value = "Update">
</form>
```

```
<%
    }
%>
</body>
</html>
```

⑫ 调用 updateById 存储过程，根据指定的主键实现更新，新建 call_update.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="java.sql.*,cap.db.*"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>Update</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));
        String name = request.getParameter("username");
        String pass = request.getParameter("password");
        String procedure="{call updateById(?,?,?)}";
        DBCon dbc = new DBCon();
        int res=dbc.doUpdate(procedure, new Object[]{name,pass,id});
        if (res > 0)
            response.sendRedirect("call_query.jsp");
        else
            out.print("更新失败");
    %>
</body>
</html>
```

⑬ 调用 addAdmin 存储过程实现向表中添加数据，新建 addadmin.jsp，addAdmin.jsp 的页面请参考本提供讲义书配套的源代码。call_insert.jsp 编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="cap.db.*"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>添加用户</title>
</head>
<body>
    <%
        String name = request.getParameter("username");
        String pass = request.getParameter("password");
        DBCon dbc = new DBCon();
        String procedure = "{call addAdmin(?,?)}";
        int res = dbc.doUpdate(procedure, new Object[]{name,pass});
        if (res > 0)
            response.sendRedirect("call_query.jsp");
        else
            out.print("添加失败");
    %>
</body>
</html>
```

⑭ 运行的结果图和 4.5 节类似，所以不再提供。

4.10　JDBC 事务

4.10.1　事务的定义

事务就是指作为单个逻辑工作单元执行的一组数据操作序列，这些操作要么必须全部成功执行，要么必须全部执行失败，以保证数据的一致性和完整性。

事务具有 ACID 属性，具体解释起来其实是指事务具有原子性、一致性、隔离性和持久性 4 个主要特性。

① 原子性(atomic)：事务由一个或多个行为绑在一起组成，好像是一个单独的工作单元。原子性确保在事务中的所有操作要么都发生，要么都不发生。

② 一致性(consistent)：一旦一个事务结束了(不管成功与否)，系统所处的状态和它的业务规则是一致的。即数据应当不会被破坏。

③ 隔离性(isolated)：事务应该允许多个用户操作同一个数据，一个用户的操作不会和其他用户的操作互相干扰。

④ 持久性(durable)：一旦事务完成，事务的结果应该持久化。

事务的 ACID 特性是由关系数据库管理系统(RDBMS)来实现的。

① 数据库管理系统采用日志来保证事务的原子性、一致性和持久性。日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。

② 数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库相同

的数据时，只允许持有锁的事务能更新该数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。

4.10.2 数据库事务声明

数据库系统的客户程序只要向数据库系统声明了一个事务，数据库系统就会自动保证事务的 ACID 特性。在 JDBC API 中，java.sql.Connection 类代表一个数据库连接。它提供了以下方法控制事务。

```
setAutoCommit(Boolean autoCommit):设置是否自动提交事务。
commit():提交事务。
rollback():撤销事务。
```

JDBC API 声明事务的示例代码如下。

```
Connection = null;
PreparedStatement pstmt = null;
try{
    con = DriverManager.getConnection(dbUrl, username, password);
    //设置手工提交事务模式
    con.setAutoCommit(false);
    pstmt = ......;
    pstmt.executeUpdate();
    //提交事务
    con.commit();
}catch(Exception e){
//事务回滚
    con.rollback();
} finally{
  .......
}
```

4.10.3 事务使用案例

本案例演示数据插入操作，设置手工提交事务模式，当插入操作出现错误，将执行事物回滚。

① 在工程 hello9 下，展开 Java Resources，右击 src 选择 New/Class，新建 JDBCTransaction.java，编辑后的代码如下。

```
package cap.trans;
import java.sql.*;
import db.DBCon;
public class JDBCTransaction {
```

```
    private static DBCon dbc  = new DBCon();
    public static void main(String[] args) throws SQLException {
        trans();
    }
    public static void trans() throws SQLException {
        Connection conn = dbc.getConnection();
        String sql = "insert into admin(username,password)" + " values(?,?)";
        PreparedStatement pstmt = null;
        try {
            pstmt = conn.prepareStatement(sql);
            conn.setAutoCommit(false);
            pstmt.setString(1, "jack");
            pstmt.setString(2, "jack");
            pstmt.executeUpdate();
            System.out.println("数据 1 插入成功");
            pstmt.setString(1, "tom");
            pstmt.setString(2, "tom");
            pstmt.executeUpdate();
            System.out.println("数据 2 插入成功");
            int i = 10/0;
            pstmt.setString(1, "tony");
            pstmt.setString(2, "tony");
            pstmt.executeUpdate();
            System.out.println("数据 3 插入成功");
            conn.commit();
        } catch (Exception e) {
            System.out.print(e.getMessage());
            conn.rollback();
        } finally {
            if (pstmt ! = null)
                pstmt.close();
            if (conn ! = null)
                conn.close();
        }
    }
}
```

代码解释：当程序执行 int i=10/0 语句时，由于分母不能为 0，将会抛出 divide by zero 错误，程序跳转到异常捕捉部分，执行回滚，之前已经执行的插入操作将会被撤销。

② 运行结果，选中 JDBCTransaction.java，右击 Run As/Java Application，由于在执行插入第三条记录的时候报错，所有事物会进行回滚，撤销原来插入的数据，最后抛出异常。

第5章

JavaBean 技术

5.1 JavaBean 的概念

JavaBean 组件，或者简称 Bean，是一种 Java 语言写成的可重用组件。它可以在一个纯 JSP 应用程序中使用 JavaBean 组件来搭建应用程序，这样可以减少 JSP 页面中逻辑的数量。在一个同时使用 Servlet 和 JSP 页面的应用程序中，Bean 可以作为数据的携带者，在这两者之间传递数据。JEE 编程模型中也推荐使用 Bean，这样可以更方便地把业务逻辑移植到 EJB 结构里，以满足用户新的需求。

JavaBean 组件只是根据一系列规定设计出来的普通的 Java 类。只有遵循这些规定，开发工具才可以知道该如何使用这个 Bean，JavaBean 规范把 Bean 定义成具有以下特性的类：

① 支持事件(event)。事件作为一种简单的通信机制，可以用来告知 Bean 许多有用的信息。

② 支持属性(property)。既可以通过工具来自定义属性，也可以在程序设计中使用属性。

③ 支持持久性(persistence)。这样就可以在应用程序开发工具中定制 Bean，然后把它的状态保存在另一个地方，而且可以在需要的时候将状态重新载入。

JavaBean 是一个类，它有一个无参数的构造函数，并且遵从 JavaBean 的命名约定。Bean 的属性可以通过获取(getter)方法和设置(setter)方法来访问，这两个方法统称为 Bean 的存取(accessor)方法。获取方法和设置方法的名称分别是由单词 get 和 set 再加上属性的名称而构成的，其中每一个单词的首字母都要大写。普通的获取方法是没有参数的，但会返回一个与属性类型相同的值，而设置方法会接受一个和属性类型相同的参数，返回的类型则是 void。一个可读的属性会有获取方法，而一个可写的属性就会有设置方法和设置方法的不同组合，一个属性就可以是只读、只写的或是可读可写的。

5.2 JavaBean 的使用案例

5.2.1 JavaBean 案例 1——实现登录

① 创建 Dynamic Web Project 工程 hello10，复制工程 hello8 下的 DBCon. java 到 cap. db 包下。工程的结构图如 5-1 所示。

② 复制 MySQL 的驱动连接到 WebContent/WEB-INF/lib 目录下。

③ 展开 Java Resources，右击 src/New/Class，出现类创建向导，输入 Package 名为 cap.

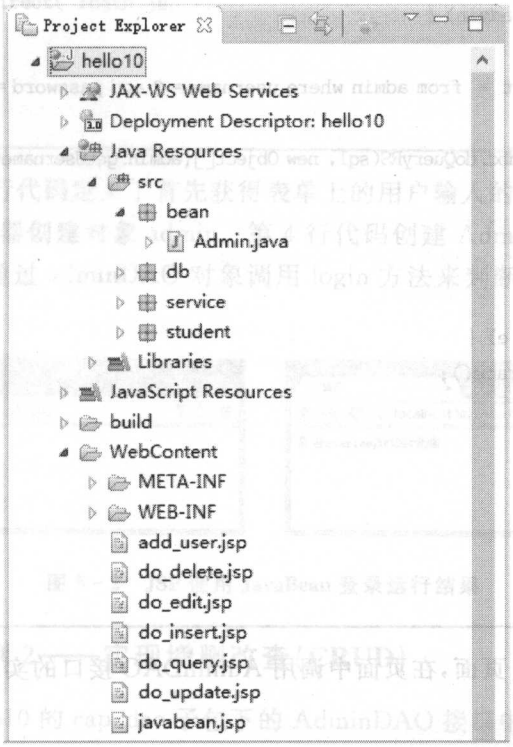


图 5-1　hello10 工程结构图

bean，类名为 Admin。创建的方法和代码详见 2.3.1。

④ 在工程 src 的 cap.dao 子包中创建 AdminDAO.java 接口，编写代码如下。

```
package cap.dao;
import java.util.ArrayList;
import cap.bean.Admin;
public interface AdminDAO {
    public int login(Admin admin);
}
```

⑤ 在工程 src 的 cap.dao.impl 子包中创建 AdminDAO.java 接口的实现类 AdminDAOImpl，编写代码如下。

```
package cap.dao.impl;
import java.sql.*;
import java.util.ArrayList;
import cap.bean.Admin;
import cap.dao.AdminDAO;
import cap.db.DBCon;
public class AdminDAOImpl implements AdminDAO {
    private DBCon dbc = null;
    @Override
```

```
    public int login(Admin admin) {
        int res = 0;
        String sql = "select * from admin where username = ? and password = ?";
        try {
            ResultSet rs = dbc.doQueryRS(sql, new Object[]{admin.getUsername(),admin.getPassword()});
            if(rs.next())
                res = 1;
            else
                res = 0;
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return res;
    }
}
```

⑥ 创建 do_login.jsp 页面，在页面中调用 AdminDAO 接口的实现 AdminDAOImpl 类来判断是否登录。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<%@page import = "cap.dao.*,cap.bean.*" %>
<%@page import = "cap.dao.impl.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>登录处理</title>
</head>
<body>
<%
    String user = request.getParameter("username");
    String pass = request.getParameter("password");
    Admin admin = new Admin(user, pass);
    AdminDAO adminDAO = new AdminDAOImpl();
    int res = adminDAO.login(admin);
    if (res = = 1){
        response.sendRedirect("index.jsp");
        session.setAttribute("u",user);
    }

    else
```

```
        response.sendRedirect("login.jsp");
%>
</body>
</html>
```

代码解释：第 1～2 行代码定义了首先获得表单上的用户输入的用户名和密码。第 3 行代码通过 Admin 类的构造器创建对象 admin。第 4 行代码创建 AdminDAOImpl 类的对象 adminDAO。第 5 行代码通过 adminDAO 对象调用 login 方法来判断是否登录成功，运行的结果如图 5－2 所示。

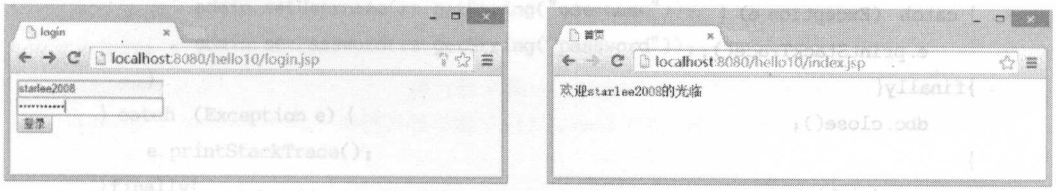


图 5－2　JSP 使用 JavaBean 登录运行结果

5.2.2　JavaBean 案例 2——实现增删改查(CRUD)

① 继续在工程 hello10 的 cap. dao 子包下的 AdminDAO 接口中添加下面的方法。

② 在 cap. dao 子包中的 AdminDAO 接口中含有下面的方法。

```
public  List<Admin> findALL() ;
public int delAdmin(int id) ;
public int addAdmin(Admin admin) ;
public  Admin findById(int id) ;
public int updateById(Admin admin) ;
```

代码解释：findAll 方法是查找表中的所有数据，返回值是 List<Admin>类型。delAdmin 方法根据传递的主键实现删除某条记录，如果成功返回大于 0 的数，失败则返回 0。addAdmin 方法根据传递的 Admin 对象 admin 朝数据库中添加一条记录，如果成功返回大于 0 的数，失败则返回 0。findById 方法根据传递的主键参数，如果查找成功返回一个 Admin 对象，如果失败返回 null。updateById 方法根据传递的 Admin 对象更新数据库中一条记录，如果成功返回大于 0 的数，失败则返回 0。

③ 在 cap. dao. imp 子包中的 AdminDAO 接口实现类 AdminDAOImpl 中添加具体的实现方法，编辑后的代码如下。

```
@Override
    public List<Admin> findALL() {
        List<Admin> adminList = new ArrayList<Admin>();
        String sql = "select * from admin order by id";
        try {
```

```
            dbc = new DBCon();
            ResultSet rs = dbc.doQuery(sql,new Object[]{});
            while(rs.next())
            {
                Admin admin = new Admin();
                admin.setId(rs.getInt("id"));
                admin.setUsername(rs.getString("username"));
                admin.setPassword(rs.getString("password"));
                adminList.add(admin);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return adminList;
    }
    @Override
    public int delAdmin(int id) {
        int res = 0;
        String sql = "delete from admin where id = ?";
        try {
            dbc = new DBCon();
            res = dbc.doUpdate(sql,new Object[]{id});
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return res;
    }
    @Override
    public int addAdmin(Admin admin) {
        int res = 0;
        String sql = "insert into admin(username,password) values(?,?)";
        try {
            dbc = new DBCon();
            res = dbc.doUpdate(sql,new Object[]{admin.getUsername(),admin.getPassword()});
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return res;
    }
```

```
    @Override
    public Admin findById(int id) {
        Admin admin = null;
        String sql = "select * from admin where id = ?";
        try {
            ResultSet rs = dbc.doQuery(sql,new Object[]{id});
            if(rs.next())
            {
                admin = new Admin();
                admin.setId(rs.getInt("id"));
                admin.setUsername(rs.getString("username"));
                admin.setPassword(rs.getString("password"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return admin;
    }
    @Override
    public int updateById(Admin admin) {
        int res = 0;
        String sql = "update admin set username = ?,password = ? where id = ?";
        try {
            res = dbc.doUpdate(sql,new Object[]{admin.getUsername(),admin.getPassword(),admin.
getId()});
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return res;
    }
```

代码解释：AdminDAOImpl 类的作用是实现数据库的增删改查。其中 findAll 方法查询数据库中的全部记录，存储到 List<Admin>对象 adminList 中并返回。delAdmin 方法根据传入的主键删除某一条记录，并返回受影响的结果集数。addAdmin 方法实现朝数据库添加记录，传入的参数为 Admin 对象 admin，返回值为受影响的结果数。findById 方法根据传入的参数主键进行查询，返回值 Admin 类对象。updateById 方法实现数据的更新，传入的参数为 Admin 对象 admin，返回值为受影响的结果。

④ 实现查询表中的所有记录，创建 do_query.jsp，编辑代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
```

```
    pageEncoding = "UTF-8" % >
< % @page import = "cap.dao. * ,java.util. * ,cap.bean. * " % >
< % @page import = "cap.dao.impl. * " % >
< ! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd" >
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8" >
<title>查询所有的记录</title>
</head>
<body>
    < %
        AdminDAO adminDAO = new AdminDAOImpl();//第 1 行代码
        List<Admin> results = adminDAO.findALL();//第 2 行代码
    % >
    <table border = "1" >
        < %
            for (int i = 0; i < results.size(); i + + ) {
                Admin admin = results.get(i);
        % >
        <tr>
            <td>< % = admin.getId() % ></td>
            <td>< % = admin.getUsername() % ></td>
            <td>< % = admin.getPassword() % ></td>
            <td><ahref = "do_delete.jsp? id = < % = admin.getId() % >" >删除</a></td>
            <td><ahref = "do_edit.jsp? id = < % = admin.getId() % >" >编辑</a></td>
        </tr>
        < %
            }
        % >
    </table>
    <ahref = "addAdmin.jsp" >添加用户</a>
</body>
</html>
```

代码解释:第 1 行代码创建 AdminDAOImpl 对象 adminDAO。第 2 行代码调用 adminDAO 对象的 findAll 方法实现查询数据库表中的所有记录,并存放在 List<Admin>对象 results 中。余下的代码通过循环读取 results 中的所有记录,并显示到页面上。

⑤ 根据主键删除指定的记录,创建 do_delete.jsp,编辑后的代码如下。

```
< % @page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" % >
< % @page import = "cap.dao. * ,java.util. * ,cap.bean. * " % >
< % @page import = "cap.dao.impl. * " % >
```

```
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>删除</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));
        AdminDAO adminDAO = new AdminDAOImpl();
        int res = adminDAO.delAdmin(id);
        if (res > 0)
        response.sendRedirect("do_query.jsp");
            else
        out.print("删除失败");
    %>
</body>
</html>
```

代码解释：第 3 行代码通过 adminDAO 对象调用 delAdmin 方法实现删除功能，传入的参数为记录的主键。

⑥ 根据主键进行查询实现编辑，创建 do_edit.jsp，编辑后代码如下。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<%@page import = "cap.dao.*,java.util.*,cap.bean.*" %>
<%@page import = "cap.dao.impl.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>编辑</title>
</head>
<body>
    <%
        int id = Integer.parseInt(request.getParameter("id"));
        AdminDAO adminDAO = new AdminDAOImpl();
        Admin admin = adminDAO.findById(id);//第 3 行代码
    %>
    <form action = "do_update.jsp"method = "post">
        <input type = "hidden"name = "id"value = "<% = admin.getId() %>">
        <input type = "text"name = "username"value = "<% = admin.getUsername() %>">
```

```
        <input type = "text"name = "password"value = "< % = admin.getPassword() % >">
        <input type = "submit"value = "更新">
    </form>
</body>
</html>
```

代码解释：第 3 行代码通过 adminDAO 对象调用 findById 方法查询指定记录，并返回 Admin 对象，传入的参数为某条记录的主键 id。

⑦ 根据指定的主键实现更新，创建 do_update.jsp，编辑后的代码如下。

```
< % @page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" % >
< % @page import = "cap.dao. * ,java.util. * ,cap.bean. * " % >
< % @page import = "cap.dao.impl. * " % >
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>更新</title>
</head>
<body>
    < %
        int id = Integer.parseInt(request.getParameter("id"));
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        Admin admin = new Admin(id, username, password);
        AdminDAO adminDAO = new AdminDAOImpl();//第 5 行代码
        int res = adminDAO.updateById(admin);//第 6 行代码
        if (res > 0)
            response.sendRedirect("do_query.jsp");
        else
            out.print("更新失败");
    % >
</body>
</html>
```

代码解释：第 5 行代码首先创建 AdminDAOImpl 对象 adminDAO，其次通过 adminDAO 对象调用 updateById 方法，传入的参数为 Admin 对象 admin。

⑧ 向指定表中添加数据，创建 addAdmin.jsp 和 do_insert.jsp，编辑后的代码分别如下。

```
< % @page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" % >
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
```

```
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>添加用户</title>
</head>
    <body>
    <form action = "do_insert.jsp" method = "post">
    <input type = "text"name = "username"><br>
    <input type = "text"name = "password"><br>
    <input type = "submit"value = "添加">
    </form>
</body>
</html>
```

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
    pageEncoding = "UTF-8" %>
<%@page import = "cap.dao.*,java.util.*,cap.bean.*" %>
<%@page import = "cap.dao.impl.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>添加用户</title>
</head>
<body>
<%
        String name = request.getParameter("username");
        String pass = request.getParameter("password");
        Admin admin = new Admin(name, pass);
        AdminDAO adminDAO = new AdminDAOImpl();
        int res = adminDAO.addAdmin(admin);//第 5 行代码
        if (res > 0)
            response.sendRedirect("do_query.jsp");
        else
            out.print("添加失败");
%>
</body>
</html>
```

代码解释：第 5 行代码通过 adminDAO 对象调用 addAmin 方法，传递的参数为 admin 对象。

⑨ 运行 do_query.jsp，运行结果如图 5-3 所示，与第 4 章类似。

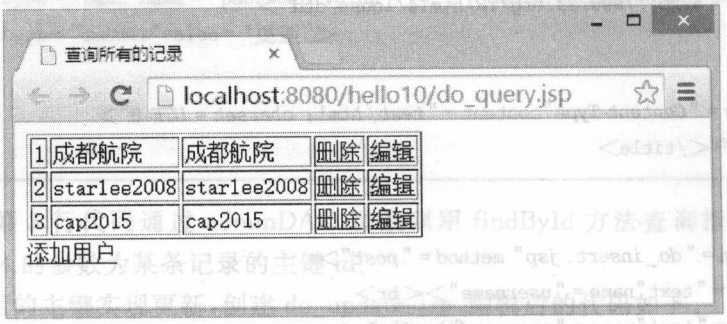


图 5-3　do_query.jsp 的运行结果

第6章 Servlet技术

6.1 Servlet技术

6.1.1 Servlet概念及生命周期

在第1章描述了Servlet的优势和问题，在本章将介绍Servlet的使用。Servlet是一种Java类，它使得服务器的功能可扩展至处理请求和生成应答。它是用Servlet API定义的类和接口实现的。API由两个程序包组成：javax.servlet程序包包含独立于协议的类和接口，而java.servlet.http程序包则提供HTTP特定的扩展和实用程序类。

Servlet的实质是实现了接口javax.servlet.Servlet的类，实现是直接完成或通过扩展某个支持类来完成的。该接口定义了Web容器用来管理Servlet和与之交互的方法。用于处理HTTP请求的Servlet一般情况下都会扩展java.servlet.http.HttpServlet类，该类实现了Servlet，并提供了适用于HTTP处理的附加方法。

Web容器管理Servlet生命周期的所有方面，它根据需要创建Servlet类的实例，将请求传递给实例进行处理，最终删除实例。对于HttpServlet来说，容器会在Servlet生命周期的适当时间调用表6-1所列的常用方法。

表6-1 HttpServlet类常用方法

序 号	方 法	描 述
1	public void init() throwe ervletExecption	方法会在发送第一个请求之前调用一次。可用于初始化Servlet的状态，例如通过读取在Web应用程序配置描述符中定义的初始化参数，或者在实例变量中保存请求处理过程中要使用的值
2	public void doGet(HttpServletRequest) request,HttpServletResponse response) Throws SerletExcepting,IOException	反复调用以使Servlet处理GET请求。Request参数提供了关于请求的详细信息，Servlet可以使用response参数生成应答
3	public void doPost(HttpServletRequest request),HttpServletResponse response) Throws ServletException,IOException	反复调用以使Servlet处理post请求

除doGet()和doPet()方法之外，还有一些对应于其他HTTP方法的方法：doDelete()、doHead()、doOptions()、doPut()和doTrace()。一般情况下不用实现这些方法，因为HttpS-

ervlet 类已经用适用于大多数，Servlet 的方法考虑到了 HEAD、OPTIONS 和 TRACE 请求，而且 DELETE 和 PUT 这两种 HTTP 方法很少用在 Web 应用程序中。

6.1.2 Servlet 案例 1——Servlet 快速入门

① 创建工程 Dynamic Web Project 工程，工程名为：hello11，其工程结构图如 6-1 所示。

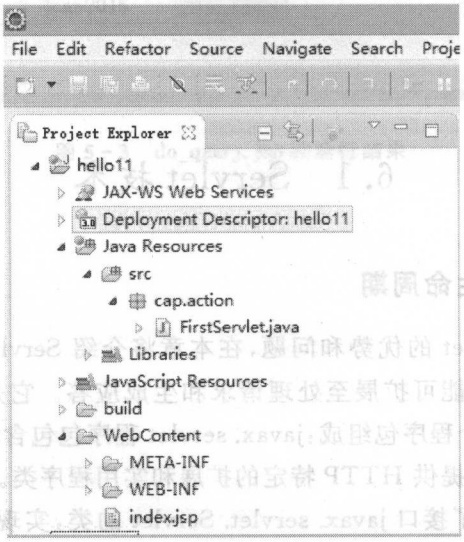


图 6-1 hello11 的工程结构图

② 展开 Java Resources，右击 src 选择 New/Servlet，首先创建第一个 Servlet 类，类名为 FirstServlet，如图 6-2 所示。

图 6-2 创建 Servlet 的向导

③ 打开刚才创建的 FirstServlet.java，编辑后的代码如下。

```
package cap.action;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FirstServlet extends HttpServlet {
private static final long serialVersionUID  = 1L;
    public FirstServlet() {        }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    PrintWriter out = response.getWriter();//第 1 行代码
    out.println("<html><head>");
    out.println("<title>First Servlet </title>");
    out.println("</head><body>");
    out.println("Hello First Servlet");
    out.println("</body></html>");
    out.close();//第 7 行代码
}
    }
```

代码解释：第 1 行代码通过 response 对象获得 PrintWriter 对象 out，然后通过 out 向页面上打印 HTML 相关标签信息。

④ 在 web.xml 进行 Servlet 的配置，配置的代码如下。

```
<? xml version = "1.0"encoding = "UTF-8"? >
<web-app xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns = "http://java.sun.com/
xml/ns/javaee" xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id = "WebApp_ID" version = "3.0">
<display-name>hello12</display-name>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
    <servlet-name>First</Servlet-name>
    <servlet-class>action.FirstServlet</Servlet-class>
</servlet>
```

```
<servlet-mapping>
    <Servlet-name>First</Servlet-name>
    <url-pattern>/first</url-pattern>
</Servlet-mapping>
</web-app>
```

配置解释：<servlet - name>为 servlet 配置一个名字 First，接着<servlet - class>指定名字为 First 的 Servlet 的类为 action. FirstServlet 类。servlet 定义好之后，接着映射到地址栏，<servlet - mapping>标签中的<Servlet - name>指定要映射的 servlet 的名字为 First，<url - pattern>标签将 First 的 servlet 映射到/first 地址。通过在浏览器中输入 http://localhost:8080/hello11/first，可以访问 FirstServlet 类。

上面的以<servlet></servlet>及<servlet - mapping></servlet - mapping>之间配置代码可以采用注解@WebServlet("/first")替代，如图 6 - 3 所示。

```
import java.io.IOException;
@WebServlet("/first")
public class FirstServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public FirstServlet() {

    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        this.doPost(request, response);

    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out = response.getWriter();
        out.println("<html><head>");
        out.println("<title>First Servlet </title>");
        out.println("</head><body>");
        out.println("Hello First Servlet");
        out.println("</body></html>");
        out.close();
    }
```

图 6 - 3　采用注解方式配置 Servlet 类

⑤ 在地址栏上输入 http://localhost:8080/hello11/first，运行的结果如图 6 - 4 所示。

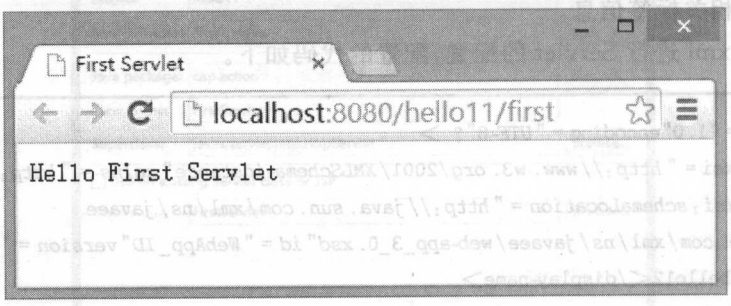


图 6 - 4　第一个 Servlet 的运行结果

6.1.3　Servlet 案例 2——Servlet 实现增删改查

① 继续在工程 hello11 中使用 Servlet 实现增删改查，工程的目录结构如图 6 - 5 所示。

② 首先复制数据库连接类 DBCon. java 到 cap. db 子包。

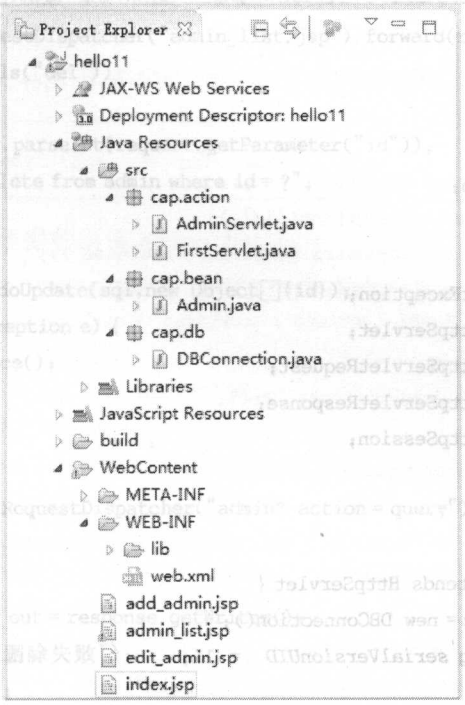


图 6-5　hello12 的工程结构图

③ 在 cap. bean 创建持久化类 Admin. java。

```
package cap.bean;
import java.io.Serializable;
public class Admin implements Serializable{
    private int id;
    private String username;
    private String password;
    public Admin() {
    }
    public Admin(int id, String username, String password) {
        this.id  = id;
        this.username  = username;
        this.password  = password;
    }
    public Admin(String username, String password) {
        this.username  = username;
        this.password  = password;
    }
    //省略 getters 和 setters
    }
```

④ 创建实现增删改查的 AdminServlet. java。

```
package cap.action;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import db.DBConnection;
import bean.Admin;
@WebServlet("/admin")
public class AdminServlet extends HttpServlet {
    private DBConnection dbc = new DBConnection();
    private static final long serialVersionUID  = 1L;
public AdminServlet() {}
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Servle-
tException, IOException {
    this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throwsServle-
tException, IOException {
        String action = request.getParameter("action");
        if(action.equals("query")){
            String sql = "select * from admin";
            ResultSet rs = null;
            List<Admin> adminList = new ArrayList<Admin>();
            try {
                rs = dbc.doQueryRS(sql,new Object[]{});
                while(rs.next())
                {
                    Admin admin = new Admin();
                    admin.setId(rs.getInt("id"));
                    admin.setUsername(rs.getString("username"));
                    admin.setPassword(rs.getString("password"));
                    adminList.add(admin);
                }
            } catch (SQLException e) {
                                e.printStackTrace();
            }
            HttpSession session = request.getSession();
```

```
        session.setAttribute("adminList", adminList);
        request.getRequestDispatcher("admin_list.jsp").forward(request, response);
    }elseif(action.equals("del"))
    {
        int id = Integer.parseInt(request.getParameter("id"));
        String sql = "delete from admin where id = ?";
        int res = 0;
        try {
            res = dbc.doUpdate(sql,new Object[]{id});
        } catch (SQLException e) {
        e.printStackTrace();
        }
        if(res>0)
        {
            request.getRequestDispatcher("admin? action = query").forward(request, response);
        }else
        {
            PrintWriter out = response.getWriter();
            out.print("删除失败");
            out.close();
        }
    }elseif(action.equals("add"))
    {
        String user = request.getParameter("username");
        String pass = request.getParameter("password");
        String sql = "insert into admin(username,password) values(?,?)";
        int res = 0;
        try {
            res = dbc.doUpdate(sql,new Object[]{user,pass});
        } catch (SQLException e) {
            e.printStackTrace();
        }
        if(res>0)
        {
            request.getRequestDispatcher("admin? action = query").forward(request, response);
        }else
        {
            PrintWriter out = response.getWriter();
            out.print("删除失败");
            out.close();
        }
    }elseif(action.equals("edit"))
    {
        int id = Integer.parseInt(request.getParameter("id"));
        String sql = "select * from admin where id = ?";
```

```
            Admin admin = null;
            try {
                ResultSet rs = dbc.doQueryRS(sql,new Object[]{id});
                if(rs.next())
                {
                    admin = new Admin();
                    admin.setId(rs.getInt("id"));
                    admin.setUsername(rs.getString("username"));
                    admin.setPassword(rs.getString("password"));
                }
            } catch (SQLException e) {
                    e.printStackTrace();
            }
            HttpSession session = request.getSession();
            session.setAttribute("admin", admin);
            request.getRequestDispatcher("edit_admin.jsp").forward(request, response);
        }elseif(action.equals("update"))
        {
            int id = Integer.parseInt(request.getParameter("id"));
            String user = request.getParameter("username");
            String pass = request.getParameter("password");
            String sql = "update admin set username = ?,password = ? where id = ?";
            int res = 0;
            try {
                res = dbc.doUpdate(sql,new Object[]{user,pass,id});
            } catch (SQLException e) {
                    e.printStackTrace();
            }
            if(res>0)
            {
                request.getRequestDispatcher("admin? action = query").forward(request, response);
            }else
            {
                PrintWriter out = response.getWriter();
                out.print("更新失败");
                out.close();
            }
        }else
        {
            System.out.println("其他方法");
        }
    }
}
```

代码解释:实现数据库操作放到 doPost()方法中实现,关键是使用页面传递过来的 action

参数进行判断。如果参数是“query”则实现查询所有的数据，如果参数为“add”则实现添加数据，根据 action 参数值实现增删改查。

⑤ 修改工程首页 index.jsp，添加一行代码，实现访问的超链接。

```
<ahref = "admin? action = query">显示所有记录</a>
```

⑥ 创建显示表中所有数据的页面 admin_list.jsp。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<%@page import = "java.util.*,cap.bean.*" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>显示所有的用户</title>
</head>
<body>
    <%
        List<Admin> adminList = (ArrayList<Admin>) session.getAttribute("adminList");
    %>
    <table border = "1">
        <%
            for (Admin admin : adminList) {
        %>
        <tr>
            <td><% = admin.getId() %></td>
            <td><% = admin.getUsername() %></td>
            <td><% = admin.getPassword() %></td>
            <td><ahref = "admin? action = del&id = <% = admin.getId() %>">删除</a></
            td>
            <td><ahref = "admin? action = edit&id = <% = admin.getId() %>">编辑</a></
            td>
        </tr>
        <%
            }
        %>
    </table>
    <ahref = "add_admin.jsp">添加用户</a>
</body>
</html>
```

⑦ 创建编辑更新的页面 edit_admin.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="bean.*"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>更新</title>
</head>
<body>
    <%
        Admin admin = (Admin) session.getAttribute("admin");
    %>
    <form action="admin?action=update"method="post">
        <input type="hidden"name="id"value="<%=admin.getId()%>"><br>
        <input type="text"name="username"value="<%=admin.getUsername()%>"><br>
        <input type="text"name="password"value="<%=admin.getPassword()%>"><br>
        <input type="submit"value="更新">
    </form>
</body>
</html>
```

⑧ 创建添加记录的页面 add_admin.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>Add Admin</title>
</head>
<body>
    <form action="admin?action=add"method="post">
        <input type="text"name="username"><br>
    <input type="text"name="password"><br>
    <input type="submit"value="添加">
    </form>
</body>
</html>
```

⑨ 打开首页 index.jsp，在如图 6-6 界面上单击“显示所有记录”，运行的结果如图 6-7

所示。

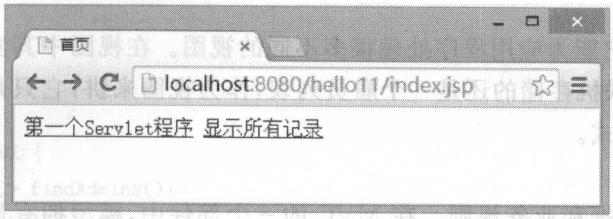


图 6-6　index.jsp 的运行结果

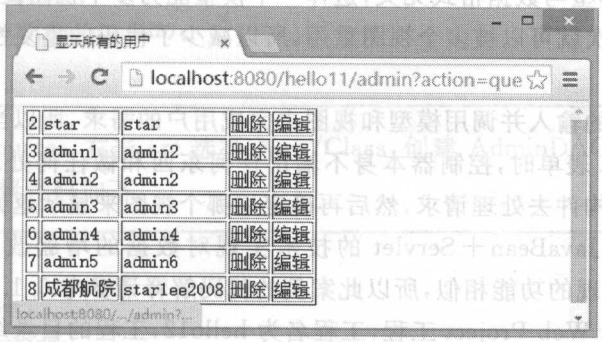


图 6-7　admin_lists.jsp 的运行结果

6.1.4　Servlet 案例 3——JSP+JavaBean+Servlet 实现增删改查

MVC 是一种软件设计框架，在 20 世纪 80 年代 Smalltalk 中出现，现已广泛使用。MVC 强制性的使应用程序的输入、处理和输出分开。其被分成三个核心部件：模型、视图和控制器，强调业务处理逻辑与用户数据显示界面的分离，使同一个程序可以呈现不同的形式。MVC 的出现提高了应用系统的可维护性、可扩展性、可移植性和组件的可复用性。最典型的 MVC 就是 JSP+Servlet+Javabean 的模式。MVC 视图如图 6-8 所示。

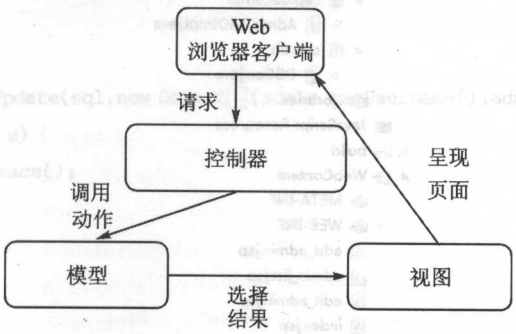


图 6-8　MVC 视图

1. 视　图

视图是用户看到并与之交互的界面。对早期的 Web 应用程序而言，视图就是由 HTML 元素组成的界面，而对现在的 Web 应用程序来说，HTML 依然发挥着重要的作用，同时一些

新的技术层出不穷，包括 Adobe Flash、像 XHTML，XML/XSL，WML 等一些标识语言和 Web services。

MVC 的优点是它能为应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

2. 模　型

模型表示企业数据和业务规则。在 MVC 的三个部件中，模型拥有最多的处理任务。如它可能用像 EJB 和 ColdFusion Components 这样的构件对象来处理数据库，被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

3. 控制器

控制器接受用户的输入并调用模型和视图去完成用户的需求，所以当单击 Web 页面中的超链接和发送 HTML 表单时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

本节的案例整合 JavaBean＋Servlet 的技术实现对数据的增删改查，与 6.1.3 节案例(Servlet 实现案例)实现的功能相似，所以此案例的代码解释请参考 6.1.3 章节。

① 创建 Dynamic Web Project 工程，工程名为 hello12，工程的目录结构如图 6－9 所示。

图 6－9　hello12 的工程结构图

② 工程中所用到的 Admin.java 和 DBCon.java 类与工程 hello12 中用到的一样，复制到本项目中如图 6－9 所示的包中。

③ 展开 Java Resources，右击 src 选择 New/Interface，创建 AdminDAO.java 接口，编辑

代码如下。

```
package cap.dao;
import java.util.List;
import cap.bean.Admin;
public interface AdminDAO {
    public List<Admin> findAdmins();
    public int addAdmin(Admin admin);
    public Admin findById(int id);
    public int delById(int id);
    public int updateAdmin(Admin admin);
}
```

④ 展开 Java Resources，右击 src 选择 New/Class，创建 AdminDAO 接口的实现类 AdminDAOImpl.java，编辑代码如下。

```
package cap.dao.impl;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import cap.bean.Admin;
import cap.dao.AdminDAO;
import cap.db.DBCon;
public class AdminDAOImpl implements AdminDAO {
    private DBCon dbc = null;
    @Override
    public int addAdmin(Admin admin){
        String sql = "insert into admin(username,password) values(?,?)";
        int res = 0;
        try {
            res = dbc.doUpdate(sql,new Object[]{admin.getUsername(),admin.getPassword()});
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return res;
    }
    @Override
    public int updateAdmin(Admin admin){
        int res = 0;
        try {
```

```
            String sql = "update admin set username = ?,password = ? where id = ?";
            res = dbc.doUpdate(sql,new Object[]{admin.getUsername(),admin.getPassword(),admin.
            getId()});
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return res;
    }
    @Override
    public List<Admin> findAdmins() {
        List<Admin> adminList = new ArrayList<Admin>();
        String sql = "select * from admin order by id";
            try {
                dbc = new DBCon();
                ResultSet rs = dbc.doQuery(sql,new Object[]{});
                while(rs.next())
                {
                    Admin u = new Admin();
                    u.setId(rs.getInt("id"));
                    u.setUsername(rs.getString("username"));
                    u.setPassword(rs.getString("password"));
                    adminList.add(u);
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }finally{
                dbc.close();
            }
            return adminList;
    }
    @Override
    public Admin findById(int id) {
        Admin admin = null;
        try {
            dbc = new DBCon();
            String sql = "select * from admin where id = ?";
            ResultSet rs = dbc.doQuery(sql,new Object[]{id});
            if(rs.next())
            {
                admin = new Admin();
                admin.setId(rs.getInt("id"));
                admin.setUsername(rs.getString("username"));
                admin.setPassword(rs.getString("password"));
```

```
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return admin;
    }
    @Override
    public int delById(int id) {
        int res = 0;
        try {
            String sql = "delete from admin where id = ?";
            res = dbc.doUpdate(sql,new Object[]{id});
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            dbc.close();
        }
        return res;
    }
}
```

代码解释：本类主要实现数据库的增删改查，读者可以根据方法名进行判断每个方法的作用，比如 updateAdmin 方法实现更新记录。

⑤ 展开 Java Resources，右击 src 选择 New/Servlet，首先创建 AdminServlet.java 类名，编辑后的代码如下所示。

```
package cap.action;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import cap.bean.Admin;
import cap.dao.AdminDAO;
import cap.dao.impl.AdminDAOImpl;
@WebServlet("/admin")
public class AdminServlet extends HttpServlet {
    private AdminDAO adminDAO = null;
```

```
    private static final long serialVersionUID  = 1L;
public AdminServlet() {
    adminDAO = new AdminDAOImpl();
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Servlet
Exception, IOException {
        this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servlet
Exception, IOException {
        request.setCharacterEncoding("utf-8");
        String action = request.getParameter("action");
        if(action.equals("query"))
        {
            List<Admin> results = adminDAO.findAdmins();
            HttpSession session = request.getSession();
            session.setAttribute("adminList", results);
            request.getRequestDispatcher("admin_list.jsp").forward(request, response);
        }elseif(action.equals("delete"))
        {
            int id = Integer.parseInt(request.getParameter("id"));
            int res = adminDAO.delById(id);
            if(res>0)
            {
                request.getRequestDispatcher("admin? action=query").forward(request, response);
            }else
            {
                PrintWriter out = response.getWriter();
                out.print("删除失败");
                out.close();
            }
        }elseif(action.equals("edit"))
        {
            int id = Integer.parseInt(request.getParameter("id"));
            Admin admin = adminDAO.findById(id);
            HttpSession session = request.getSession();
            session.setAttribute("admin", admin);
            request.getRequestDispatcher("edit_admin.jsp").forward(request, response);
        }elseif(action.equals("update"))
        {
            int id = Integer.parseInt(request.getParameter("id"));
            String username = new String(request.getParameter("username"));
            String password = new String(request.getParameter("password"));
            Admin admin = new Admin(id,username,password);
            int res = adminDAO.updateAdmin(admin);
```

```
            if(res>0)
            {
                request.getRequestDispatcher("admin? action=query").forward(request, response);
            }else
            {
                PrintWriter out=response.getWriter();
                out.print("更新失败");
                out.close();
            }
        }elseif(action.equals("add"))
        {
            String username=request.getParameter("username");
            String password=request.getParameter("password");
            Admin admin=new Admin(username,password);
            int res=adminDAO.addAdmin(admin);
            if(res>0)
            {
                request.getRequestDispatcher("admin? action=query").forward(request, response);
            }else
            {
                PrintWriter out=response.getWriter();
                out.print("删除失败");
                out.close();
            }
        }
    }
}
```

代码解释：调用 AdminDAO 类的方法实现具体的数据库操作和页面跳转。

6.1.5　Servlet 案例 4——JSP＋JavaBean＋Servlet 实现分页

本案例采用完整的 MVC 分层结构实现分页显示，是 5.6 节（脚本实现数据库分页显示）的进一步完善。

① 打开 Eclipse 新建 Dynamic Web Project，工程名为 hello13，创建好的工程的目录结构如图 6-10 所示。

② 复制数据库连接类 DBCon.java 到 cap.db 包中，复制 Admin.java 到 cap.bean 包下。

③ 在 cap.util 子包下创建分页类 PageBean.java，编辑代码如下。

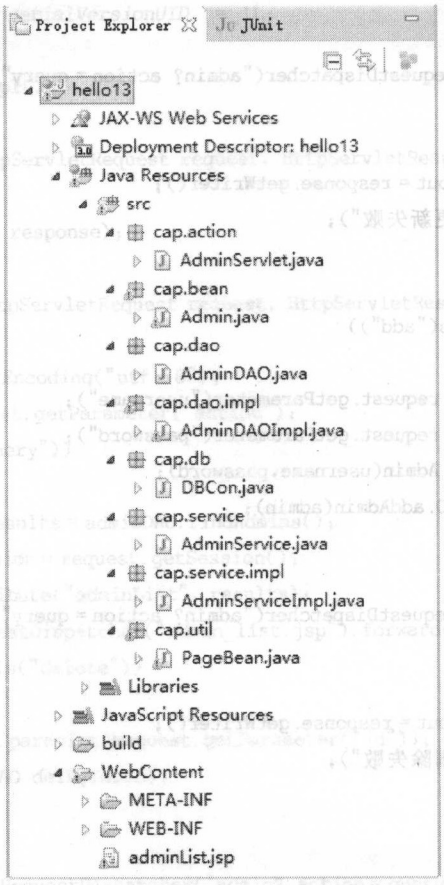


图 6-10　hello13 的工程结构图

```
package cap.util;
import java.util.List;
public class PageBean {
    //结果集
    private List list;
    //每页多少条数据
    private int pageSize;
    //第几页
    private int pageNo;
    //查询总记录数
    private int totalRecords;

    /**
     * 计算总的页面数
     * @return
     */
    public int getTotalPages(){
```

```
        if(totalRecords % pageSize = = 0){
            returntotalRecords/pageSize;
        }else
        {
            returntotalRecords/pageSize + 1;
        }
    }
    /* *
     * 取得首页
     * @return
     */
    public int getTopPageNo(){
        return 1;
    }
    /* *
     * 上一页
     * @return
     */
    public int getPreviousPageNo(){
        if(pageNo< = 1){
            return 1;
        }
        returnpageNo-1;

    }
    /* *
     * 取得尾页
     * @return
     */
    public int getBottomPageNo(){
        return getTotalPages();
    }
    /* *
     * 下一页
     * @return
     */
    public int getNextPageNo(){
        if(pageNo> = getBottomPageNo())
            return getBottomPageNo();
        returnpageNo + 1;


    }
    public List getList() {
        returnlist;
    }
```

```
    public void setList(List list) {
        this.list  = list;
    }
    public int getPageSize() {
        returnpageSize;
    }
    public void setPageSize(int pageSize) {
        this.pageSize  = pageSize;
    }
    public int getPageNo() {
        returnpageNo;
    }
    public void setPageNo(int pageNo) {
        this.pageNo  = pageNo;
    }
    public int getTotalRecords() {
        returntotalRecords;
    }
    public void setTotalRecords(int totalRecords) {
        this.totalRecords  = totalRecords;
    }
}
```

④ 创建数据访问对象(DAO)接口:AdminDAO.java,编辑代码如下。

```
package cap.dao;
import java.util.List;
import cap.bean.Admin;
public interface AdminDAO {
    public List<Admin> findByPage(int pageNo,int pageSize);
    public int getTotalRecords();
}
```

⑤ 创建实现 AdminDAO 接口的类:AdminDAOImpl.java,编辑代码如下。

```
package cap.dao.impl;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import cap.bean.Admin;
import cap.dao.AdminDAO;
import cap.db.DBCon;
public class AdminDAOImpl implements AdminDAO {
    private DBCon dbc = null;
```

```
    @Override
    public List<Admin> findByPage(int pageNo, int pageSize) {
        List<Admin> adminList = new ArrayList<Admin>();
        try {
            dbc = new DBCon();
            int start = (pageNo-1) * pageSize;
            String sql = "select * from admin limit ?,?";
            ResultSet rs = dbc.doQuery(sql, new Object[]{start,pageSize});
            while(rs.next()){
                Admin admin = new Admin();
                admin.setId(rs.getInt("id"));
                admin.setUsername(rs.getString("username"));
                admin.setPassword(rs.getString("password"));
                adminList.add(admin);
            }
        } catch (Exception e) {
            e.printStackTrace();

        }finally{
            dbc.close();
        }
        return adminList;
    }
    @Override
    public int getTotalRecords() {
        int count = 0;
        try {
            dbc = new DBCon();
            String sql = "select count(*) as t from admin";
            ResultSet rs = dbc.doQuery(sql, new Object[]{});
            if(rs.next()){
                //count = rs.getInt(1);对应于没有 as t
                count = rs.getInt("t");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return count;
    }
}
```

代码解释：findByPage 方法主要功能是实现分页，传递的参数为需要显示的当前页 pageNo 和每页显示的记录数 pageSize，最后返回的结果是查询 pageNo 页面的结果。

⑥ 创建服务层接口：AdminService.java，编辑代码如下。

```
package cap.service;
import cap.util.PageBean;
public interface AdminService {
    public PageBean findByPage(int pageNo,int pageSize);
}
```

⑦ 创建实现 AdminService 接口的类：AdminServiceImpl.java。

```
package cap.service.impl;
import java.util.List;
import cap.bean.Admin;
import cap.dao.AdminDAO;
import cap.dao.impl.AdminDAOImpl;
import cap.service.AdminService;
import cap.util.PageBean;

public class AdminServiceImpl implements AdminService {
    private AdminDAO adminDAO = new AdminDAOImpl();

    @Override
    public PageBean findByPage(int pageNo, int pageSize) {
        PageBean pg = new PageBean();
        List<Admin> adminList = adminDAO.findByPage(pageNo, pageSize);
        pg.setList(adminList);
        pg.setPageNo(pageNo);
        pg.setPageSize(pageSize);
        pg.setTotalRecords(adminDAO.getTotalRecords());
        return pg;
    }
}
```

代码解释：在 findByPage 方法中，首先创建一个 PageBean 的对象 pg，其次通过 adminDAO 对象的 findByPage 获取分页的数据，最后将 pageNo、pageSize、分页的结果集、总的页面数存储到 PageBean 对象 pg 中。

⑧ 在 cap.action 包下创建 AdminServlet 类，编辑后的代码如下。

```
package cap.action;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import cap.service.AdminService;
import cap.service.impl.AdminServiceImpl;
import cap.util.PageBean;
@WebServlet("/admin.html")
public class AdminServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private AdminService adminService = null;
    private int pageNo;

    public AdminServlet() {
    adminService = new AdminServiceImpl();
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String action = request.getParameter("action");
        if(action.equals("pager")){
            //实现分页
            String pageNoStr = request.getParameter("pageNo");
            if(pageNoStr = = null){
                pageNo = 1;
            }else
            {
                pageNo = Integer.parseInt(pageNoStr);
            }
            PageBean pg = adminService.findByPage(pageNo, 5);
            request.getSession().setAttribute("pg", pg);
            request.getRequestDispatcher("adminList.jsp").forward(request, response);
        }
    }
}
```

代码解释：将 AdminServlet 类通过注解@WebServlet("/admin.html")映射到地址栏上，通过 http://localhost:8080/hello13/admin.html?action=pager 地址访问实现分页的显示。

⑨ 页面的调用，创建 adminList.jsp，编辑后的代码如下。

```
<%@page import = "cap.bean.Admin" %>
<%@page import = "java.util.List" %>
<%@page import = "cap.util.PageBean" %>
<%@page language = "java" contentType = "text/html; charset = utf-8"
```

```
pageEncoding = "utf-8" % >
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type" content = "text/html; charset = utf-8">
<title>分页显示</title>
</head>
<body>
< %
    PageBean pg = (PageBean)session.getAttribute("pg");
    List<Admin> adminList = pg.getList();
% >
<table>
< % for(Admin admin:adminList){ % >
    <tr>
    <td>< % = admin.getId() % ></td>
    <td>< % = admin.getUsername() % ></td>
    <td>< % = admin.getPassword() % ></td>
    </tr>
< % } % >
    <tr><tdcolspan = "5">
    共  < % = pg.getTotalPages() % > 页     
    当前第  < % = pg.getPageNo() % >页      
    <ahref = "admin.html? action = pager&pageNo = < % = pg.getTopPageNo() % >">首页</a>
    <ahref = "admin.html? action = pager&pageNo = < % = pg.getPreviousPageNo() % >">上一页</a>
    <ahref = "admin.html? action = pager&pageNo = < % = pg.getNextPageNo() % >">下一页</a>
    <ahref = "admin.html? action = pager&pageNo = < % = pg.getBottomPageNo() % >">尾页</a>
    </td></tr>

</table>
</body>
</html>
```

代码解释：第 1～2 行代码通过 session 获得分页的数据，第 3 行代码同样通过 session 获得分页条 pgBar。

⑩ 运行的结果如图 6-11 所示。

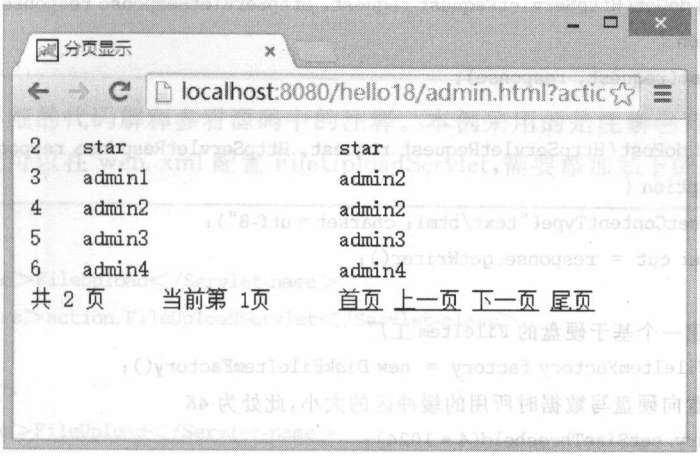


图 6-11　JSP 分层实现分页的效果

6.1.6　Servlet 案例 5——文件上传

① 文件上传使用的是 Apache 下的 commons 组件来实现，包括 commons-fileupload-1.3.jar 和 commons-io-2.4.jar 两个包。详见工程 hello14 中的 lib 目录。

② 新建工程 hello14，在其 src 下的 cap.util 子包新建 Servlet，名称为 FileUploadServlet.java。

```
package cap.util;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

@WebServlet("/upload")
public class FileUploadServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private String filePath = "e:\\upload";//存放上传文件的目录
    private String tempFilePath = "e:\\upload\\temp";//存放临时文件的目录
    public FileUploadServlet() {
    }
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletEx-
ception, IOException {
        this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle-
tException, IOException {
        response.setContentType("text/html; charset = utf-8");
        PrintWriter out = response.getWriter();
           try {
            //创建一个基于硬盘的 FileItem 工厂
            DiskFileItemFactory factory = new DiskFileItemFactory();
            //设置向硬盘写数据时所用的缓冲区的大小,此处为 4K
            factory.setSizeThreshold(4 * 1024);
            //设置临时目录
            factory.setRepository(new File(tempFilePath));
            //创建一个文件上传处理器
            ServletFileUpload upload = new ServletFileUpload(factory);
            //设置允许上传的文件的最大尺寸,此处为 4M
            upload.setSizeMax(4 * 1024 * 1024);
            List items = upload.parseRequest(request);
            Iterator iter = items.iterator();
            while(iter.hasNext()) {
             FileItem item = (FileItem)iter.next();
             if(item.isFormField()) {
             String name = item.getFieldName();
                String value = item.getString();
                out.println(name + ":" + value + "\r\n");
             } else {
                //上传文件,此处实现文件上传
               String filename = item.getName();
                    int index = filename.lastIndexOf("\\");
                    filename = filename.substring(index + 1, filename.length());
                    long fileSize = item.getSize();
                    if(filename.equals("") || fileSize = = 0)
                     return;
                    File uploadFile = new File(filePath + "/" + filename);
                    item.write(uploadFile);
                    out.println(filename + "已经保存");
                    out.println("文件 " + filename + " 的大小: " + fileSize + "\r\n");
              }
             }
             out.close();
            } catch (Exception ex) {
             ex.printStackTrace();
            }
```

```
    }
}
```

代码解释：详细的代码解释参看源码中的注释。本例采用的是注解@WebServlet 的方式进行配置，当然也可以在 web. xml 配置 FileUploadServlet，需要添加如下的代码块。

```
<Servlet>
    <Servlet-name>FileUpload</Servlet-name>
    <Servlet-class>action. FileUploadServlet</Servlet-class>
</Servlet>
<Servlet-mapping>
    <Servlet-name>FileUpload</Servlet-name>
    <url-pattern>/upload</url-pattern>
</Servlet-mapping>
```

③ 新建文件上传页面 upload. jsp。

```
<%@page language = "java"contentType = "text/html; charset = UTF-8"
pageEncoding = "UTF-8" %>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>文件上传</title>
</head>
<body>
<body>
    <form method = "POST"action = "upload"ENCTYPE = "multipart/form-data">
        文件 1:<input name = "x"size = "40"type = "file"><br>
        文件 2:<input name = "y"size = "40"type = "file"><br>
        文件 3:<input name = "z"size = "40"type = "file"><br>
        <input name = "upload"type = "submit"value = "开始上传"/>
    </form>
</body>
</body>
</html>
```

⑤ 运行的结果如图 6 - 12 所示。

注：上传之前需要创建存放上传文件存放临时文件的目录，在本案例中是需要在 Windows 操作系统的 E 盘下创建一个 upload 的目录，然后在 upload 中再创建一个 temp 的子目录，这样才能上传成功。

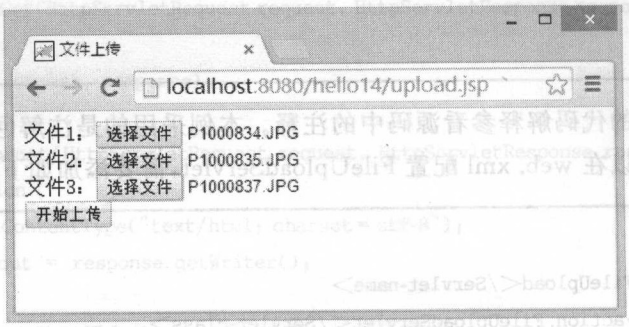


图 6-12　文件上传运行图

6.1.7　Servlet 案例 6——邮件发送

① 接着上面工程 hello14，新建 SendMailServlet.java，编辑代码如下。

```
package cap.util;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.mail.SimpleEmail;
@WebServlet("/mail")
public class SendMailServlet extends HttpServlet {
private static final long serialVersionUID  = 1L;
public SendMailServlet() {     }
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    this.doPost(request, response);
    }
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html; charset=UTF-8");
    SimpleEmail mail=new SimpleEmail();
    mail.setHostName("smtp.yeah.net");
    mail.setAuthentication("starlee_cdavtc", "l12345678");
    try{
        mail.setCharset("utf-8");
        mail.addTo(request.getParameter("to"));
        mail.setFrom(request.getParameter("from"));
        mail.setSubject(request.getParameter("subject"));
        mail.setMsg(request.getParameter("body"));
```

```
            mail.send();
            request.getSession().setAttribute("msg", "邮件发送成功");

        }catch(Exception e)
        {
            request.getSession().setAttribute("msg", "邮件发送失败");
            System.out.print(e.getMessage());
        }
        request.getRequestDispatcher("/result.jsp").forward(request, response);
    }
}
```

② 创建发送邮件的页面 email.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>发送邮件</title>
</head>
<body>
    <form action="mail"method="post">
        收件人<input type="text"name="to"size="30"><br>
        发件人<input type="text"name="from"size="30"><br>
        主题<input type="text"name="subject"size="30"><br>
        正文<textarea cols="50"rows="15"name="body"></textarea><br>
        <input type="submit"value="提交">
</form>
</body>
</html>
```

③ 创建发送结果页面 result.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>发送结果</title>
</head>
```

```
<body>
    <% = session.getAttribute("msg") %>
</body>
</html>
```

④ 运行的结果请读者参考源码自行验证。

6.1.8 Servlet 案例 7——验证码

现在的网站普遍都添加了验证码进行验证，以保证系统的安全。下面接着讲解验证码的生成以及验证。

① 接着上面工程 hello14，在 cap.util 新建 AuthCode.java，编辑代码如下。

```
package cap.util;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.util.Random;
public class AuthCode {
    public static final int AUTHCODE_LENGTH  = 5;  //验证码长度  02.
    public static final int SINGLECODE_WIDTH  = 15;  //单个验证码宽度  03.
    public static final int SINGLECODE_HEIGHT  = 30; //单个验证码高度  04.
    public static final int SINGLECODE_GAP  = 4;  //单个验证码之间间隔  05.
    public static final int IMG_WIDTH = AUTHCODE_LENGTH * (SINGLECODE_WIDTH + SINGLECODE_GAP);
    public static final int IMG_HEIGHT  = SINGLECODE_HEIGHT;
    public static String getAuthCode() {
    String authCode = "";
    for(int i = 0; i <AUTHCODE_LENGTH; i + + ) {
    authCode + = (new Random()).nextInt(10);
    }
    return authCode;
    }
    public static BufferedImage getAuthImg(String authCode){
    BufferedImage img = new BufferedImage(IMG_WIDTH, IMG_HEIGHT, BufferedImage.TYPE_INT_BGR);
    //得到图片上的一个画笔
    Graphics g = img.getGraphics();
    //设置画笔的颜色，用来做背景色
    g.setColor(Color.YELLOW);
    //用画笔来填充一个矩形，矩形的左上角坐标，宽，高
    g.fillRect(0, 0, IMG_WIDTH, IMG_HEIGHT);
    //将画笔颜色设置为黑色，用来写字
    g.setColor(Color.BLACK);
    //设置字体：宋体、不带格式的、字号
    g.setFont(new Font("宋体", Font.PLAIN, SINGLECODE_HEIGHT  + 5));
```

```
    //输出数字
    char c;
    for(int i = 0; i < authCode.toCharArray().length; i++) {
    //取到对应位置的字符
    c = authCode.charAt(i);
    //画出一个字符串:要画的内容,开始的位置,高度
    g.drawString(c + "", i * (SINGLECODE_WIDTH + SINGLECODE_GAP) + SINGLECODE_GAP / 2, IMG_
HEIGHT);
    }
    Random random = new Random();
    //干扰素
    for(int i = 0; i < 20; i++) {
    int x = random.nextInt(IMG_WIDTH);
    int y = random.nextInt(IMG_HEIGHT);
    int x2 = random.nextInt(IMG_WIDTH);
    int y2 = random.nextInt(IMG_HEIGHT);
    g.drawLine(x, y, x + x2, y + y2);
    }
return img;
}
}
```

② 在 cap.util 新建 AuthCodeServlet.java。

```
package cap.util;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/code")
public class AuthCodeServlet extends HttpServlet {
private static final long serialVersionUID = 1L;

public AuthCodeServlet() {
}
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletExcep-
tion, IOException {

String scode = AuthCode.getAuthCode();
request.getSession().setAttribute("scode", scode);
//将验证码保存到 session 中,便于以后验证
try{
//发送图片
```

```
        ImageIO.write(AuthCode.getAuthImg(scode), "JPEG", response.getOutputStream());
    } catch(IOException e){
        e.printStackTrace();
    }

    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    }
    }
```

③ 新建验证码的显示和验证页面 code.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>验证码</title>
</head>
<body>
<script type="text/javascript">
    function fresh(){
    var timestamp = new Date();
    document.getElementById("code").src="code? time="+timestamp;
    }
</script>
    <form action="verify.jsp"method="post">
    <img src="code"id="code"/><ahref="#"onclick="fresh()">看不清</a>
    <input type="text"name="ucode"><br>
    <input type="submit"value="提交">
    </form>
</body>
</html>
```

④ 新建验证页面 verify.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>验证结果</title>
</head>
<body>
<%
    String scode = (String)session.getAttribute("scode");
    String ucode = request.getParameter("ucode");
    if(scode.equals(ucode)){
    out.println("输入的验证码正确");
    }else
    {
    out.println("输入的验证码错误");
    }
%>
</body>
</html>
```

⑤ 运行的结果如图 6-13 所示，读者可以输入正确和错误的验证码进行验证。

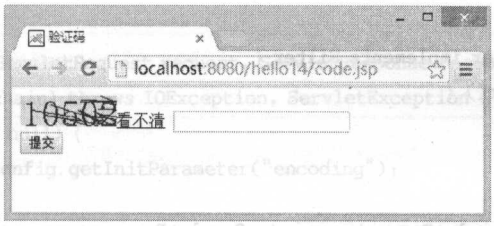


图 6-13　验证码的显示

6.2　Servlet 过滤器(Filter)

6.2.1　过滤器

过滤器是一个 Java 组件，它先于与之相关的 Servlet 或 JSP 页面运行在服务器上。过滤器可附加到一个或多个 Servlet 或 JSP 页面上，并且可以检查进入这些资源的请求信息。这样可以很容易地将应用于所有请求的任务集中在一起，例如访问控制、登录和内容的开销或应用程序提供服务等。在这之后，过滤器可以选择以何种方式调用资源，比如以常规的方式调用资源(即，调用 Servlet 或 JSP 页面)；利用修改过的请求信息调用资源；阻止该资源调用，代之以转到其他的资源，返回一个特定的状态代码或生成替换输出等。Servlet 作为过滤器使用时，它可以对客户的请求进行处理。处理完成后，它会交给下一个过滤器处理，这样，客户的请求在过滤链里逐个处理，直到请求发送到目标为止。

在 JSP 应用中通过使用 Servlet 过滤器进行编码转换，就是将编码转换为 UFT-8。可以很好的解决 Web 程序的国际化问题，即在 Web 应用程序中使用 UTF-8 对字符进行编码。

这样做的好处在于不仅解决了 Web 上中文字符编码问题，而且其他所有的字符编码也都统一使用 UTF-8，实现了语言的国际化。同时也解决了保存数据到数据库时需编码转换的问题。

6.2.2 Servlet 过滤器案例 1——登录拦截、字符转化

本案例采用 Servlet 来实现对中文乱码的解决，将字符转化为 UTF-8。

① 在工程 hello14，展开 Java Resources，右击 src 选择 New/Servlet，创建 CheckLoginFilter.java，编辑代码如下。

```
package cap.util;
import java.io.IOException;
import javax.Servlet.Filter;
import javax.Servlet.FilterChain;
import javax.Servlet.FilterConfig;
import javax.Servlet.ServletException;
import javax.Servlet.ServletRequest;
import javax.Servlet.ServletResponse;
import javax.Servlet.annotation.WebFilter;
import javax.Servlet.http.HttpServletRequest;
import javax.Servlet.http.HttpServletResponse;
@WebFilter("/*")
public class CheckLoginFilter implements Filter {
    public CheckLoginFilter() {
    }
    public void destroy() {
    }
    public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;
    String uriStr = req.getRequestURI();
    boolean flag = true;
    if (uriStr.indexOf("login") == -1
    && req.getSession().getAttribute("u") == null)
        flag = false;
    if (flag)
    chain.doFilter(request, response);
    else {
    res.sendRedirect("login.jsp");
    }
    }
    public void init(FilterConfig fConfig) throws ServletException {
    }
}
```

代码解释：在 doFilter()方法中，首先获得用户请求的 URL，接着判断如果用户的 URL

没包含 login.jsp 页面并且 session 中的 u 值为空，则将 flag 标志设置为 false，如果 flag 标志为 false，则跳转到登录页面，否则，留给下一个 Filter 进行处理。

② 新建 EncodingFilter.java，编辑代码如下。

```
package cap.util;
import java.io.IOException;
import javax.Servlet.Filter;
import javax.Servlet.FilterChain;
import javax.Servlet.FilterConfig;
import javax.Servlet.ServletException;
import javax.Servlet.ServletRequest;
import javax.Servlet.ServletResponse;
import javax.Servlet.annotation.WebFilter;
@WebFilter(urlPatterns = "/*",initParams = @WebInitParam(name = "encoding",value = "utf-8"))
public class EncodingFilter implements Filter {
    private String encoding = null;
    private FilterConfig config;
    public EncodingFilter() {
    }
    public void destroy() {
    }
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        if (encoding == null) {
            encoding = config.getInitParameter("encoding");
        }
        if (encoding != null) {
            request.setCharacterEncoding(encoding);
            response.setContentType("text/html;charset=" + encoding);
        }
        chain.doFilter(request, response);
    }
    public void init(FilterConfig fConfig) throws ServletException {
        this.config = fConfig;
    }
}
```

代码解释：doFilter()方法中，首先判断 encoding 是否为 null，如果为 null 的话，则采用 FilterConfig 的 config 对象读取编码，encoding 编码在配置文件 web.xml 中定义。其次通过 request 对象将编码设置为 encoding 的值，也就是 utf-8，response 对象设置页面的类型和字符编码，最后将请求传到下一个过滤器处理。

③ 本案例采用@WebFilter 注解实现，当然也可以采用修改 web.xml 实现，具体实现如下代码块所示。

```
<filter>
<filter-name>CheckLogin</filter-name>
<filter-class>action.CheckLoginFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>CheckLogin</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
<filter>
<filter-name>EncodingFilter</filter-name>
<filter-class>action.EncodingFilter</filter-class>
<init-param>
<param-name>encoding</param-name>
<param-value>utf-8</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>EncodingFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

④ 新建首页页面 index.jsp。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>首页</title>
</head>
<body>
<%=(String)session.getAttribute("u")%>
</body>
</html>
```

⑤ 在工程 WebContent 中新建 login.jsp 页面，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv = "Content-Type"content = "text/html; charset = UTF-8">
<title>登录</title>
</head>

<body>
    <form action = "do_login.jsp" method = "post">
        <p>用户:<input type = "text"name = "username">
        <p>密码:<input type = "text"name = "password">
        <p><input type = "submit"value = "登录"><input type = "reset"value = "重置"/></p>
    </form>
</body>
</html>
```

⑥ 新建处理登录的 LoginServlet.java 类,编辑后的代码如下。

```
package cap.action;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public LoginServlet() {
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        String name = request.getParameter("username");
        String pass = request.getParameter("password");
        if(name.equals("成航院")&&pass.equals("成航院")){
            request.getSession().setAttribute("u", name);
            request.getRequestDispatcher("index.jsp").forward(request, response);
        }else
        {
            request.getRequestDispatcher("login.jsp").forward(request, response);
        }
    }
}
```

⑦ 本案例首先采用 Servlet 过滤器 EncodingFilter 进行字符转化，其次使用 CheckLoginFilter 过滤器进行判断是否登录，如果登录成功，则跳转到 index. jsp，否则跳转到 login. jsp 页面。

6.3 Servlet 监听器(Listener)

6.3.1 监听器

JSP/Servlet 中的事件通过实现 Listener 接口的类可以在特定事件(Event)发生时，呼叫特定的方法来对事件进行响应，由 Container 在特定事件发生时呼叫特定的实现 Listener 的类。在 JSP /Servlet 中，共有 8 个 Listener 接口，6 个 Event 类别。

1. ServletContextListener 接口

接口方法：contextInitialized()与 contextDestroyed()。

接收事件：ServletContextEvent。

触发场景：在 Container 加载 Web 应用程序时(例如启动 Container 之后)，会呼叫 contextInitialized()，而当容器移除 Web 应用程序时，会呼叫 contextDestroyed ()方法。

2. ServletContextAttributeListener

接口方法：attributeAdded()、attributeReplaced()、attributeRemoved()。

接收事件：ServletContextAttributeEvent。

触发场景：若有对象加入为 application(ServletContext)对象的属性，则会呼叫 attributeAdded()，同理在置换属性与移除属性时，会分别呼叫 attributeReplaced()、attributeRemoved()。

3. HttpSessionListener

接口方法：sessionCreated()与 sessionDestroyed ()。

接收事件：HttpSessionEvent。

触发场景：在 session(HttpSession)对象建立或被消灭时，会分别呼叫这两个方法。

4. HttpSessionAttributeListener

接口方法：attributeAdded()、attributeReplaced()、attributeRemoved()。

接收事件：HttpSessionBindingEvent

触发场景：若有对象加入为 session(HttpSession)对象的属性，则会呼叫 attributeAdded()，同理在置换属性与移除属性时，会分别呼叫 attributeReplaced()、attributeRemoved()。

5. HttpSessionActivationListener

接口方法：sessionDidActivate()与 sessionWillPassivate()。

接收事件：HttpSessionEvent。

触发场景：Activate 与 Passivate 是用于置换对象的动作，当 session 对象为了资源利用或负载平衡等原因而必须暂时储存至硬盘或其他储存器时(透过对象序列化)，所作的动作被称之为 Passivate，而硬盘或储存器上的 session 对象重新加载 JVM 时所作的动作被称之为 Activate。

6. ServletRequestListener

接口方法：requestInitialized()与 requestDestroyed()。

接收事件：RequestEvent。

触发场景：在 request(HttpServletRequest)对象建立或被消灭时，会分别呼叫这两个方法。

7. ServletRequestAttributeListener

接口方法：attributeAdded()、attributeReplaced()、attributeRemoved()。

接收事件：HttpSessionBindingEvent。

触发场景：若有对象加入为 request(HttpServletRequest)对象的属性，则会呼叫 attributeAdded()，同理在置换属性与移除属性时，会分别呼叫 attributeReplaced()、attributeRemoved()。

8. HttpSessionBindingListener

接口方法：valueBound()与 valueUnbound()。

接收事件：HttpSessionBindingEvent。

触发场景：实现 HttpSessionBindingListener 接口的类别，其实例如果被加入至 session(HttpSession)对象的属性中，则会呼叫 valueBound()，如果被从 session(HttpSession)对象的属性中移除，则会呼叫 valueUnbound()，实现 HttpSessionBindingListener 接口的类别不需在 web. xml 中设定。

6.3.2　Lisenter 使用案例

本案例通过实现 HttpSessionAttributeListener 接口来监听在线用户人数，当有新用户登录的时候，会触发 attributeAdded()方法。

① 在 Eclipse 中创建 hello15 工程，工程结构图如图 6-14 所示。

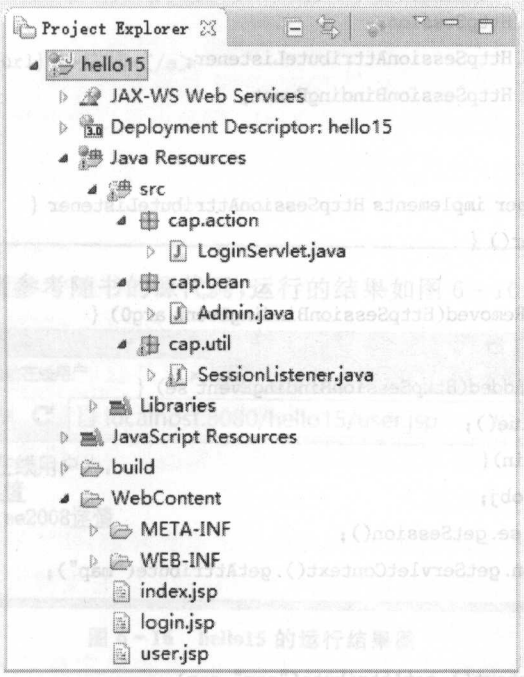


图 6-14　helo15 工程结构图

② 在 cap. util 中创建 Listener，如图 6－15 所示：类名为：SessionListener。编辑后的代码如下。

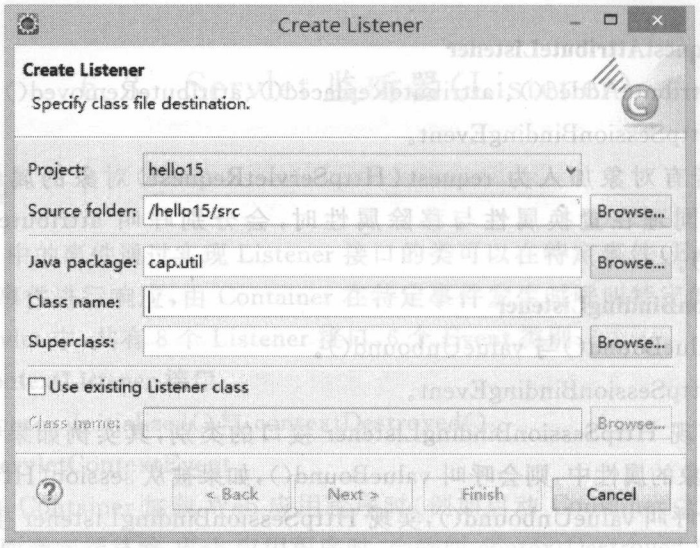


图 6－15　Listener 创建向导

```
package cap.util;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;
import cap.bean.Admin;
@WebListener
public class SessionListener implements HttpSessionAttributeListener {
    public SessionListener() {
    }
    public void attributeRemoved(HttpSessionBindingEvent arg0) {
    }
    public void attributeAdded(HttpSessionBindingEvent se) {
    Object obj = se.getValue();
    if(obj instanceof Admin){
    Admin admin = (Admin) obj;
    HttpSession session = se.getSession();
    Map map = (Map) session.getServletContext().getAttribute("map");
    if(map = = null){
    map = new HashMap();
    session.getServletContext().setAttribute("map",map);
    }
```

```
        map.put(admin.getUsername(),session);
        }
        }
        public void attributeReplaced(HttpSessionBindingEvent arg0) {
        }
}
```

③ 创建显示在线用户列表的页面 user.jsp，编辑后的代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core"prefix="c"%>
<! DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>当前在线用户</title>
</head>
<body>
当前在线用户为:
<br/>
  <c:forEachitems="${map}"var="me">
      <c:urlvalue="/kick"var="url">
          <c:paramname="username"value="${me.key}"></c:param>
      </c:url>
  ${me.key}<ahref="${url}">详情</a>
      <br/>
  </c:forEach>
</body>
</html>
```

④ 余下的代码请读者参考随书的源代码，运行的结果如图 6-16 所示。

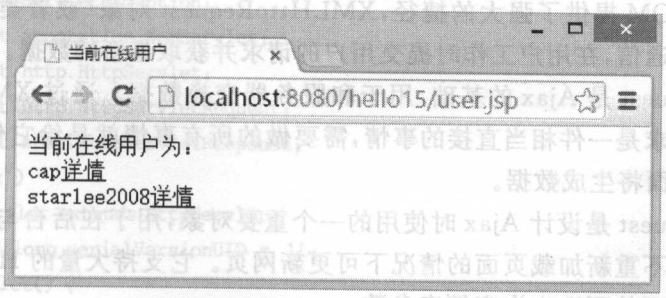


图 6-16　hello15 的运行结果图

第7章

Ajax技术

7.1 Ajax概念

异步JavaScript和XML(Asynchronous JavaScript and XML,Ajax)是一种在无须重新加载整个网页的情况下,能够更新部分网页的技术。Ajax不是单一的技术,而是四种技术的集合。

① JavaScript:JavaScript是通用的脚本语言,用来嵌入在某种应用之中,Web浏览器中嵌入的JavaScript解释器允许通过程序与浏览器的很多内建功能进行交互,Ajax应用程序是使用JavaScript编写的。

② CSS(层叠样式表):CSS为Web页面元素提供了一种可重用的可视化样式的定义方法,它提供了简单而又强大的方法,以一致的方式定义和使用可视化样式,在Ajax应用中,用户界面的样式可以通过CSS独立修改。

③ DOM(文档对象模型):DOM以一组可以使用JavaScript操作的可编程对象展现出Web页面的结构,通过使用脚本修改DOM。Ajax应用程序可以在运行时改变用户界面,或者高效地重绘页面中的某个部分。

④ XMLHttpRequest对象:XMLHttpRequest对象允许Web程序员从Web服务器以后台活动的方式获取数据。数据格式通常是XML,但是也可以很好地支持任何基于文本的数据格式。尽管XMLHttpRequest对于完成这件工作来说是最为灵活和通用的工具,但还有其他方法也可以从服务器获取数据。

通过使用JavaScript操作DOM来改变和刷新用户界面,不断地重绘和重新组织显示给用户的数据,并且处理用户基于鼠标和键盘的交互。CSS为应用提供了一致的外观,并且为以编程方式操作DOM提供了强大的捷径,XMLHttpRequest对象(或者类似的机制)则用来与服务器进行异步通信,在用户工作时提交用户的请求并获取最新的数据。

XMLHttpRequest是Ajax的基础,用于和服务器交换数据。通过XMLHttpRequest对象向服务器发送请求是一件相当直接的事情,需要做的所有事情就是给它传递一个服务器页面的URL,这个页面将生成数据。

XMLHttpRequest是设计Ajax时使用的一个重要对象,用于在后台与服务器交换数据。它最大的优点是在不重新加载页面的情况下可更新网页。它支持大量的HTTP调用语义,包括用来动态生成页面的可选查询字符串参数。

下面通过一个案例讲解使用XMLHttpRequest对象来实现Ajax,本案例中要使用到Eclipse的Javascript插件Spket,可以方便编写JavaScript的代码,详细使用参见附录C。

7.2　XMLHttpRequest 实现 Ajax

本案例通过使用 XMLHttpRequest 对象来实现 Ajax 的异步数据的返回。

① 创建工程 Dynamic Web Project 工程，工程名为：hello16，创建好的工程结构图如图 7－1 所示。

图 7－1　hello16 工程结构图

② 展开 Java Resources，右击 src 选择 New/Servlet，首先创建 AjaxServlet.java，编辑代码如下。

```
package action;
import java.io.IOException;
import java.io.PrintWriter;
import javax.Servlet.ServletException;
import javax.Servlet.annotation.WebServlet;
import javax.Servlet.http.HttpServlet;
import javax.Servlet.http.HttpServletRequest;
import javax.Servlet.http.HttpServletResponse;
@WebServlet("/ajax")
public class AjaxServlet extends HttpServlet {
private static final long serialVersionUID = 1L;
    public AjaxServlet() {
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletEx-
    ception, IOException {
```

```
    this.doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle-
    tException, IOException {
        response.setContentType("text/html;charset=utf-8");
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String str = username + "的密码是:" + password;
        System.out.println(str);
        PrintWriter out = response.getWriter();
        out.println(str);
        out.close();
    }
}
```

代码解释:doPost()方法中,首先通过 response 对象设置字符编码,接着通过 request 对象获得用户名和密码,并将用户名和密码组成字符串 str,最后通过 PrintWriter 对象 out 将 str 字符串打印到页面。

③ 在 WebContent 下新建 ajax.jsp,编辑代码如下。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>ajax 使用案例 1</title>
</head>
<script type="text/javascript">
var xmlHttp;
function createXMLHttpRequest()
{
    if(window.ActiveXObject)
    {
      xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    elseif(window.XMLHttpRequest)
    {
      xmlHttp=new XMLHttpRequest();
    }
}
function createQueryString()
{
```

```
    var username = document.getElementById("username").value;
    var password = document.getElementById("password").value;
    var queryString = "username = " + username + "&" + "password = " + password;
    return queryString;
}
function sendGet()
{
    createXMLHttpRequest();
    var queryString = "ajax?";
    queryString = queryString + createQueryString() + "&timeStamp = " + new Date().getTime();
     xmlHttp.onreadystatechange = handleStateChange;
     xmlHttp.open("GET",queryString,true);
     xmlHttp.send(null);
}
function sendPost()
{
    createXMLHttpRequest();
    var url = "ajax? timeStamp = " + new Date().getTime();
    var queryString = createQueryString();
     xmlHttp.open("POST",url,true);
     xmlHttp.onreadystatechange = handleStateChange;
     xmlHttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
     xmlHttp.send(queryString);
}
function handleStateChange()
{
if(xmlHttp.readyState = = 4)
 {
if(xmlHttp.status = = 200)
  {
   parseResults();
  }
 }
}
function parseResults()
{
    alert(xmlHttp.responseText);
}
</script>
<body>
<form method = "post"action = "">
        用户名:<input name = "username"type = "text"id = "username"/><br>
        密码:<input name = "password"type = "text"id = "password"/><br>
        <input type = "button"value = "get"onclick = "sendGet();"/>
        <input type = "button"value = "post"onclick = "sendPost();"/>
```

```
    </form>
</body>
</html>
```

代码解释：createXMLHttpRequest()方法根据不同的浏览器初始化 xmlHttp 对象，确保在微软的 IE 或者 Mozilla Firefox 之类的浏览器上都能运行。sendGet()方法采用 get 方法提交数据，首先调用 createXMLHttpRequest()方法创建 xmlHttp 对象，其次创建要发送的数据字符串，其中调用了 createQueryString()方法，sendGet()方法采用 post 方法提交数据。

④ 运行的结果如图 7-2 所示。

图 7-2　调用 Servlet 实现 ajax 的效果

7.3　jQuery 实现 Ajax

7.3.1　jQuery

互联网的发展推动了 Web 页面的快速发展。如今的 Web 页面功能已经非常强大了，而人们的需求也在提升，他们更多地关注页面展示形式和用户体验度。JavaScript 语言以其自身的语言优势不仅能满足程序开发者的需求，还能帮助程序开发者开发出用户体验度很高的页面，因而其被关注度也越来越高。jQuery 是一个优秀的 JavaScript 框架，代码高效、兼容性强等优点使得它备受广大程序员的推崇。

jQuery 是由美国人 John Resig 于 2006 年创建的一个开源项目，随着逐步的应用，越来越多的程序爱好者参与进来，完善并壮大其项目内容。如今它已发展成为集 JavaScript、CSS、DOM 和 Ajax 于一体的强大框架体系。它的应用宗旨是：以更少的代码，实现更多的功能。

jQuery 的功能主要有以下 5 点。

① 访问和操作 DOM 元素。jQuery 提供了丰富的 DOM 操作方法。使用 jQuery 库，可以很方便地获取和修改页面中的某元素，无论是删除、移动还是复制某元素，jQuery 都提供了一整套方便、快捷的方法，既减少了代码的编写，又大大提高了用户对页面的体验度。

② 控制页面样式。通过引入 jQuery，程序开发人员可以便捷方便地控制页面的 CSS 文件。使用 jQuery 操作页面样式使其能够在不同浏览器中兼容，同时还尽量修复了一些浏览器

之间的差异。

③ 对页面事件的处理。引入 jQuery 库后，可以实现脚本与页面的分离，开发者更多地专注于程序的逻辑与功效，页面设计者侧重于页面的优化与用户体验。然后，通过事件绑定机制，可以很轻松地实现两者的结合。

④ 丰富的插件运用。基于 jQuery 开发的插件已有数千个，同时还在发展中。使用这些插件可丰富完善页面功效，如利用插件来实现制表、字段提示、动画等任务，扩展页面展示效果也可轻松实现了。

⑤ 与 Ajax 技术的完美结合。Ajax 的异步读取服务器数据的方法，提高了程序开发效率，加深了用户的页面体验度；而引入 jQuery 库后，通过使用其内部对象或函数，加上几行代码就可以实现复杂的功能，即以最少的代码做最多的事情。

7.3.2　jQuery 相关操作

jQuery 封装了大量常用的 DOM 操作，对 DOM 的操作能力，是其非常重要的内容。jQuery 提供一系列与 DOM 相关的方法，这使访问和操作元素和属性变得很容易。DOM 定义访问 HTML 和 XML 文档的标准是：W3C 文档对象模型独立于平台和语言的界面，允许程序和脚本动态访问和更新文档的内容、结构以及样式。

3 个用于 DOM 操作的简单实用的 jQuery 方法。

表 7-1　3 个用于 DOM 操作的简单实用的 jQuery 方法

序　号	方　法	描　述
1	text()	设置或返回所选元素的文本内容
2	html()	设置或返回所选元素的内容(包括 HTML 标记)
3	val()	设置或返回表单字段的值

Ajax 技术最终体现在与服务器的动态数据实现人机交互中，即客户端传递带有参数的请求给服务器，服务器接收后分析所传递来的请求，并做出相应的响应。发送响应数据至客户端，客户端接收请求返回的数据，从而实现了数据的双向传递。交互式函数如表 7-2 所列。

表 7-2　jQuery 的 Ajax 交互式方法

序　号	方　法	描　述
1	$.get()	通过调用全局函数 $.get()实现了 xml 文档的加载。除加载数据外，$.get 函数还可以实现数据的请求
2	$.post()	带参数向服务器发出数据请求。全局函数 $.post()与 $.get()在本质上没有太大的区别，所不同的是，GET 方式不适合传递数据量较大的数据。同时，其请求的历史信息会保存在浏览器的缓存中，有一定的安全风险。而以 POST 方式向服务器发送数据请求，则不存在这两个方面的不足
3	$.ajax()	在 jQuery 中，还有一个功能更为全面的最底层方法 $.ajax()，该方法不仅可以很方便地实现上述三个全局函数完成的功能，还更多地关注实现过程中的细节。读者可以查阅相关文档进行更深入的学习

7.3.3 JSON 概念

JSON 简单说就是 Javascript 中的对象和数组，所以这两种结构就是对象和数组两种结构，通过这两种结构可以表示各种复杂的结构。

① 对象：对象在 js 中表示为“{}”括起来的内容，数据结构为 {key:value,key:value,...} 的键值对的结构。在面向对象的语言中，key 为对象的属性，value 为对应的属性值，所以很容易理解，取值方法为“对象.key”获取属性值，这个属性值的类型可以是数字、字符串、数组、对象几种。

② 数组：数组在 js 中是中括号“[]”括起来的内容，数据结构为 ["Java","Javascript","vb",...]，取值方式和所有语言中一样，使用索引获取，字段值的类型可以是数字、字符串、数组、对象几种。

JSON 可以将 JavaScript 对象中表示的一组数据转换为字符串，然后就可以在函数之间轻松地传递这个字符串，或者在异步应用程序中将字符串从 Web 客户机传递给服务器端程序。这个字符串看起来有点儿古怪，但是 JavaScript 很容易解释它，而且 JSON 可以表示比“名称/值”对更复杂的结构。如，可以表示数组和复杂的对象，而不仅仅是键和值的简单列表。

按照最简单的形式，可以用下面这样的 JSON 表示“名称/值对”。

```
{"id":1,"password":"cdap","username":"cdap"}
```

从语法方面来看，这与“名称/值对”相比并没有很大的优势，但是在这种情况下 JSON 更容易使用，而且可读性更好。

当需要表示一组值时，JSON 不但能够提高可读性，而且可以减少复杂性。下面的代码就是使用带花括号的记录分组在一起。

```
[{"id":1,"password":"cdap","username":"cdap"},
{"id":2,"password":"cdavtc","username":"cdavtc"},
{"id":3,"password":"成都航院","username":"成都航院"}]
```

在处理 JSON 格式的数据时，没有需要遵守的预定义的约束。所以，在同样的数据结构中，可以改变表示数据的方式，甚至可以以不同方式表示同一事物。

利用这样的语法，可以处理任何 JSON 格式的数据，而不需要使用任何额外的 JavaScrip 工具包或 API。

7.3.4 jQuery 使用案例

本案例利用 jQuery 库实现 Ajax 的数据的异步返回，实现和 7.2 节相同的功能，可以看到使用 jQuery 精简了 JS 代码。

① 在浏览器的地址栏输入“jquery.com”，进入官方网站，单击“Download jQuery v1.10 or v2.0.3”，单击“Download the uncompressed development jQuery 2.0.3”，将 js 另存。

② 在 hello16/WebContent/New/Folder 目录下新建一个文件夹，出现如图 7-3 的对话框，在对话框的 Folde name 文本框中输入“js”，单击“Finish”完成“js”文件夹的创建。

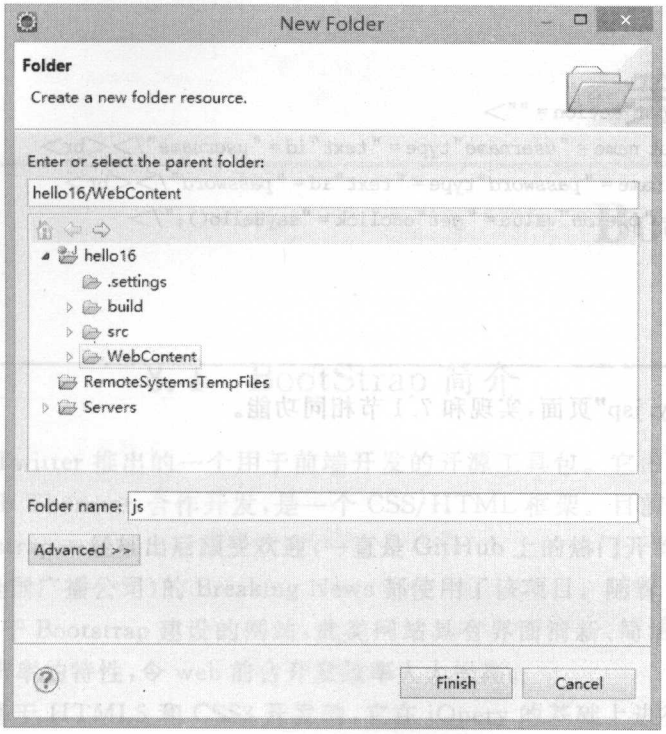


图 7-3　Folder 创建对话框

③ 将下载好的 jQuery 文件重命名为“jquery.js”，复制到新建的 js 文件夹中。

④ 在工程 hello16 下 WebContent 目录中创建“jquery.jsp”页面。

```
<%@page language="java"contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8">
<title>ajax 使用案例 2</title>
<script type="text/javascript"src="js/jquery.js"></script>
</head>
<script type="text/javascript">
function sayHello()
{
    var username=$("#username").val();
    var password=$("#password").val();
    $.post("ajax",{"username":username,"password":password},function(data)
        {
            alert(data);
        });
```

```
}
</script>
<body>
  <form method = "post"action = "">
      用户名:<input name = "username"type = "text"id = "username"/><br>
      密码:<input name = "password"type = "text"id = "password"/><br>
      <input type = "button"value = "get"onclick = "sayHello();"/>
  </form>
</body>
</html>
```

⑤ 运行“jquery.jsp”页面,实现和 7.1 节相同功能。

第8章 BootStrap3

8.1 BootStrap 简介

Bootstrap 是 Twitter 推出的一个用于前端开发的开源工具包。它由 Twitter 的设计师 Mark Otto 和 Jacob Thornton 合作开发，是一个 CSS/HTML 框架。目前，Bootstrap 最新版本为 3.3.1。Bootstrap 一经推出后颇受欢迎，一直是 GitHub 上的热门开源项目，包括 NASA 的 MSNBC(微软全国广播公司)的 Breaking News 都使用了该项目。随着 BootStrap 的推广，同时涌现了许多基于 Bootstrap 建设的网站，此类网站具有界面清新、简洁，要素排版灵活大方，自适应屏幕分辨率的特性，令 web 前台开发效率大大提高。

Bootstrap 是基于 HTML5 和 CSS3 开发的，它在 jQuery 的基础上进行了更为个性化和人性化的完善，形成一套自己独有的网站风格，并兼容大部分 jQuery 插件。

Bootstrap 中包含了丰富的 Web 组件，根据这些组件，可以快速的搭建一个漂亮、功能完备的网站。其中包括以下组件：下拉菜单、按钮组、按钮下拉菜单、导航、导航条、面包屑、分页、排版、缩略图、警告对话框、进度条和媒体对象等。

Bootstrap 自带了 13 个 jQuery 插件，这些插件使得 Bootstrap 中的组件更加丰富多彩，其中包括：模式对话框、标签页、滚动条、弹出框等。

Bootstrap 要求 html5 的文件类型，所以必须在每个使用 bootstrap 页面的开头添加如下语句。

```
<! DOCTYPE html>
<html>
<head lang = "en">
</html>
```

8.2 使用 WebStorm 开发页面

① 首先到 WebStorm 的官方网站上下载 WebStorm，下载的地址是：http://download.jetbrains.com/webstorm/WebStorm-10.0.2.exe。

② 打开安装好的 WebStorm，出现如图 8-1 所示的界面。

③ 选择 Create New Project，创建 Empty 工程，工程名为 BootStrap3，如图 8-2 所示。

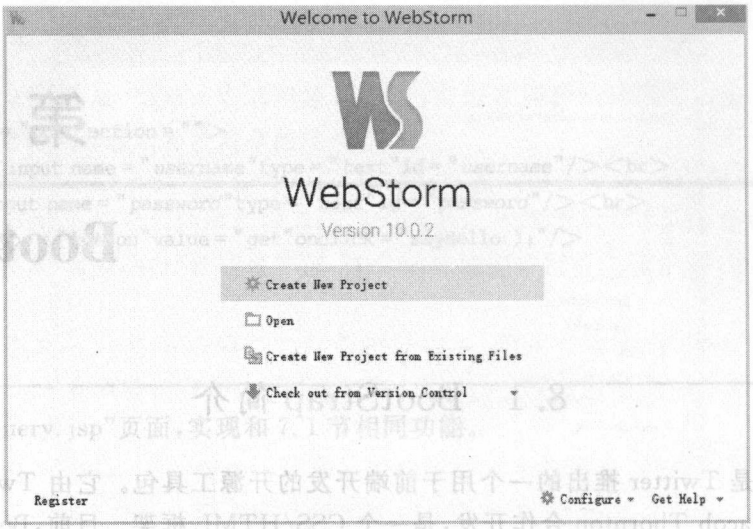


图 8-1　WebStorm 界面

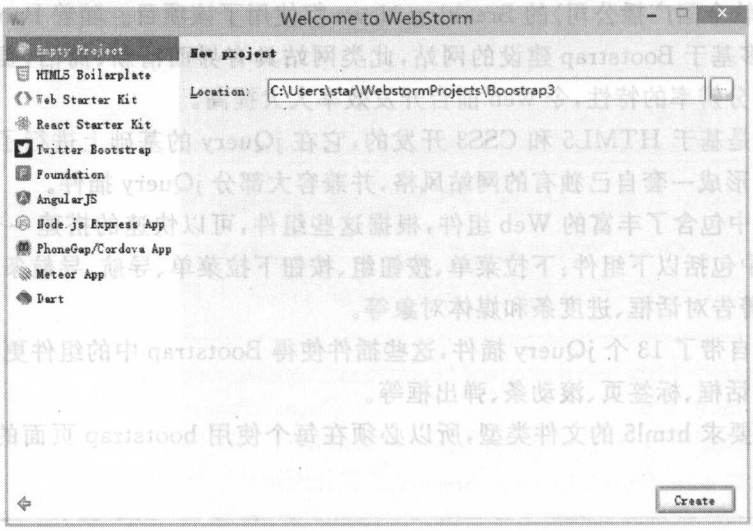


图 8-2　创建 BootStrap3 工程示意图

④ 到 BootStrap 的官网上下载 Bootstrap，下载地址为 http://getbootstrap.com/，下载完成之后将 bootstrap-3.x.x-dist.zip 解压到当前目录下，双击进入之后会看见如图 8-3 的目录结构。

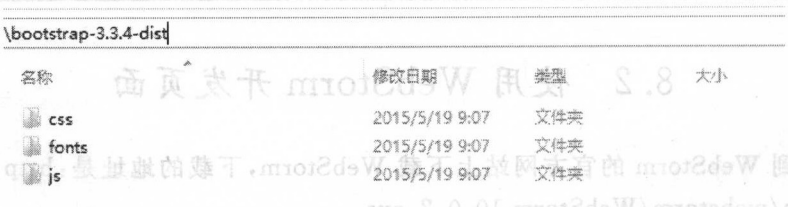


图 8-3　目录结构

⑤ 在 bootstrap3 工程中新建 Directory，命名为 bootstrap，然后将 css、fonts 和 js 复制到此目录下，同时将 jquery.js 拷贝到 js 目录中。工程的结构目录如图 8-4 所示。

⑥ 在 b3 目录中新建 html5 页面，创建的步骤如图 8－5 所示。

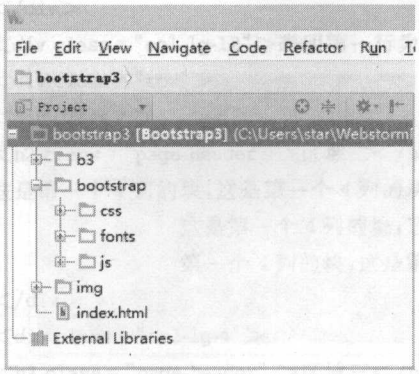


图 8－4　bootstrap3 目录结构

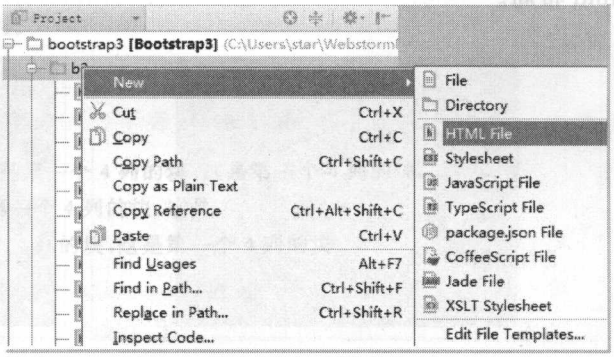


图 8－5　b3 目录中新建 html5 页面

⑦ 输入文件名为 boot00，生成后的页面代码下。

```
<! DOCTYPE html>
<html>
<head lang = "en">
<meta charset = "UTF-8">
<title></title>
</head>
    <body>
    </body>
</html>
```

8.3　BootStrap 布局

Bootstrap 建立了一个响应式的 12 列网格布局系统，提供一个宽 940 像素，12 列的格网，能够使得网页可以更好地适应多种终端设备(PC，平板电脑，智能手机等)。如图 8－6 所示。

图 8－6　BootStrap 网格布局示意图

它引入了 fixed 和 fluid-with 两种布局方式，如图 8-7 所示。左边为 Fixed 布局，右边为 Fluid 布局。

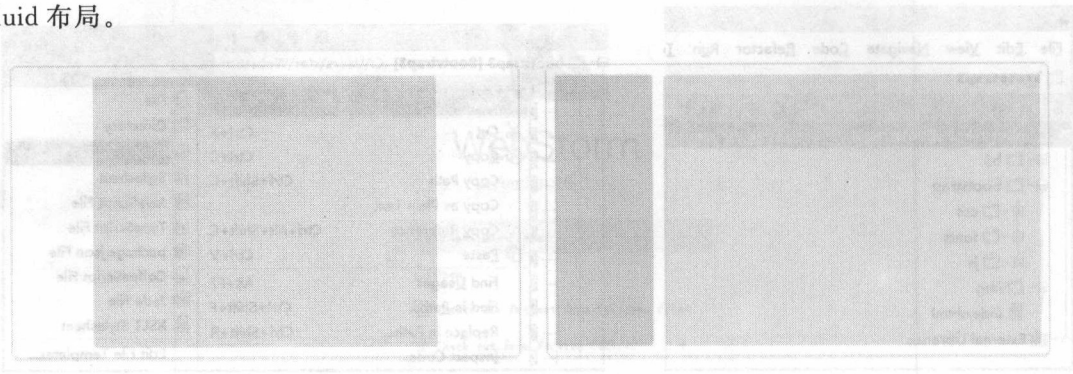


图 8-7　Fixed 布局和 Fluid 布局

8.3.1　固定布局

在 bootstrap 中，默认的布局为固定布局，图 8-8 是一个固定布局案例效果。

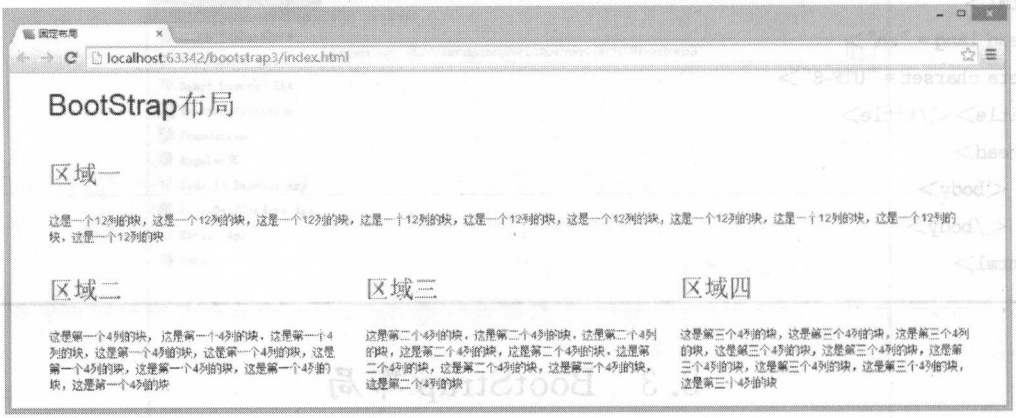


图 8-8　固定布局效果图

图 8-8 中，屏幕分成了 2 行，第一行为 12 列，第二行为 3 个 4 列的区域，container 这个类设置了宽度，还可以让内容显示在页面的中间。

具体设计的代码片段如下。

```
<body>
<div class = "container">

<div class = "row">
<div class = "col-lg-12">
<h2 class = "page-header" >区域一</h2>
这是一个 12 列的块，这是一个 12 列的块，这是一个 12 列的块，这是一个 12 列的块，这是
        一个 12 列的块，这是一个 12 列的块，这是一个 12 列的
        块，这是一个 12 列的块，这是一个 12 列的块，这是一个 12 列的块
```

```
</div>
</div>
<div class = "col-lg-12">表明第一行为 12 列。
<div class = "row">
<div class = "col-lg-4">
<h2 class = "page-header" >区域二</h2>
这是第一个 4 列的块,这是第一个 4 列的块,这是第一个 4 列的块,这是第一个 4 列的块,
        这是第一个 4 列的块,这是第一个 4 列的块,这是
        第一个 4 列的块,这是第一个 4 列的块,这是第一个 4 列的块
</div>
<div class = "col-lg-4">
<h2 class = "page-header" >区域三</h2>
这是第二个 4 列的块,这是第二个 4 列的块,这是第二个 4 列的块,这是第二个 4 列的块,
        这是第二个 4 列的块,这是第二个 4 列的块,这是第二个 4 列的块,这是
        第二个 4 列的块,这是第二个 4 列的块
</div>
<div class = "col-lg-4">
<h2 class = "page-header" >区域四</h2>
这是第三个 4 列的块,这是第三个 4 列的块,这是第三个 4 列的块,这是第三个 4 列的块,
        这是第三个 4 列的块,这是第三个 4 列的块,这是第三个 4 列的块,这是
        第三个 4 列的块,这是第三个 4 列的块
</div>
<div class = "col-lg-4">表明第二行由三个 4 列的区域组成

</div>
</div>
</body>
```

8.3.2　流式布局

如果要使用流式布局,只需要将实现固定布局中的代码 修改为 , 修改为 ,其布局将充满整个屏幕。完成后的效果如图 8-9 所示。

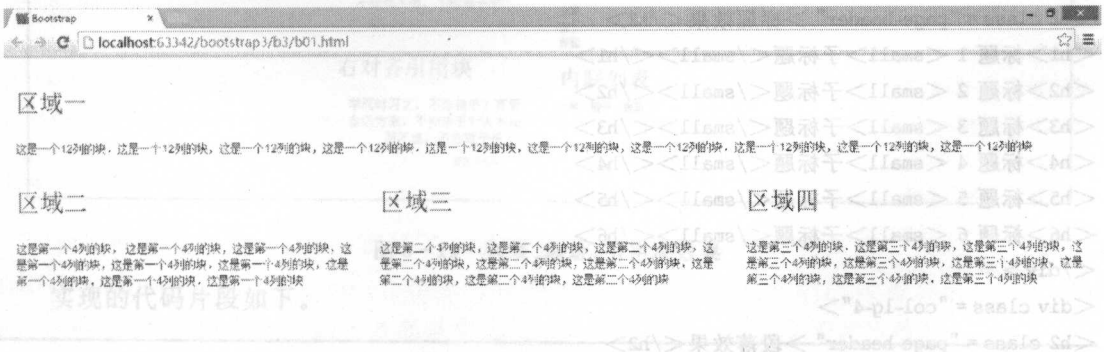


图 8-9　流式布局效果图

8.4 BootStrap 页面元素

BootStrap 的基础 CSS 提供了一系列基础的 HTML 页面要素，可以帮助开发者实现一致的页面外观，包括排版(Typography)、表格(Table)、表单(Forms)和按钮(Buttons)这四个方面的内容。

8.4.1 排 版

BootStrap 的排版，包括了标题(Headings)、段落 (paragraphs)、列表(lists)、文字(text)以及其他内联元素。用户可以通过修改 variables.less 的两个变量：@baseFontSize 和@base-LineHeight 来控制整体排版的样式。Bootstrap 同时还使用其他一些方法来创建所有类型元素的 margin、padding、line-height 等。标题、段落和文字效果如图 8-10 所示。

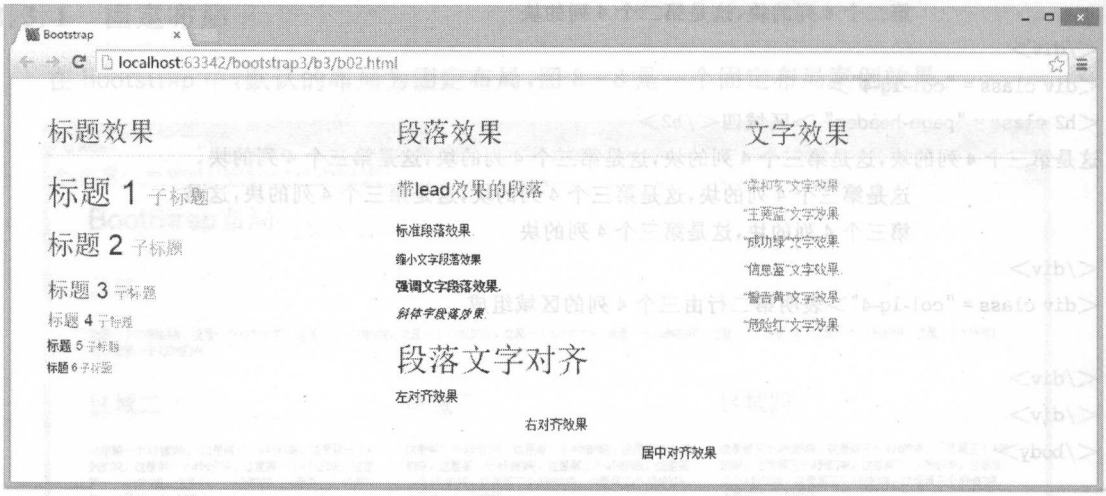


图 8-10 标题、段落、文字效果图

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-4">
<h2 class = "page-header" >标题效果</h2>
<h1>标题 1 <small>子标题</small></h1>
<h2>标题 2 <small>子标题</small></h2>
<h3>标题 3 <small>子标题</small></h3>
<h4>标题 4 <small>子标题</small></h4>
<h5>标题 5 <small>子标题</small></h5>
<h6>标题 6 <small>子标题</small></h6>
</div>
<div class = "col-lg-4">
<h2 class = "page-header" >段落效果</h2>
<p class = "lead">带 lead 效果的段落</p>
```

```
<p>标准段落效果.</p>
<p><small>缩小文字段落效果</small></p>
<p><strong>强调文字段落效果.</strong></p>
<p><em>斜体字段落效果.</em></p>
<h1>段落文字对齐</h1>
<p class = "text-left">左对齐效果</p>
<p class = "text-center">右对齐效果</p>
<p class = "text-right">居中对齐效果</p>
</div>
<div class = "col-lg-4">
<h2 class = "page-header" >文字效果</h2>
<p class = "text-muted">“柔和灰”文字效果</p>
<p class = "text-primary">“主要蓝”文字效果</p>
<p class = "text-success">“成功绿”文字效果</p>
<p class = "text-info">“信息蓝”文字效果.</p>
<p class = "text-warning">“警告黄”文字效果</p>
<p class = "text-danger">“危险红”文字效果</p>
</div>
</div>
```

Bootstrap 提供三种标签来表现不同类型的列表。<ul>表示无序列表，<ul class="unstyled">表示无样式的无序列表，<ol>表示有序列表，<dl>表示描述列表，<dl class="dl-horizontal">表示竖排描述列表。

此外，在 BootStrap 中，可以自动的进行引用块的样式设置。如图 8-11 所示。

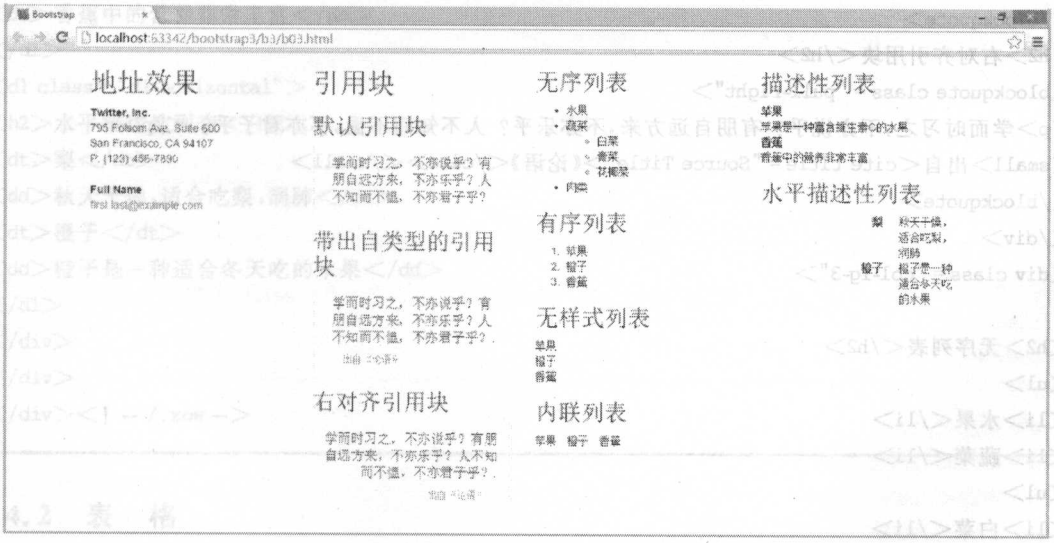


图 8-11　列表、地址、引用块效果

实现的代码片段如下。

```
<div class = "row">
```

```
<div class = "col-lg-3">

<h1>地址效果</h1>
<address>
<strong>Twitter, Inc.</strong><br>
795 Folsom Ave, Suite 600<br>
San Francisco, CA 94107<br>
<abbr title = "Phone">P:</abbr> (123) 456-7890
</address>

<address>
<strong>Full Name</strong><br>
<a href = "mailto:#">first.last@example.com</a>
</address>
</div>
<div class = "col-lg-3">
<h1>引用块</h1>
<h2>默认引用块</h2>
<blockquote>
<p>学而时习之,不亦说乎? 有朋自远方来,不亦乐乎? 人不知而不愠,不亦君子乎? </p>
</blockquote>
<h2>带出自类型的引用块</h2>
<blockquote>
<p>学而时习之,不亦说乎? 有朋自远方来,不亦乐乎? 人不知而不愠,不亦君子乎?.</p>
<small>出自<cite title = "Source Title">《论语》</cite></small>
</blockquote>
<h2>右对齐引用块</h2>
<blockquote class = "pull-right">
<p>学而时习之,不亦说乎? 有朋自远方来,不亦乐乎? 人不知而不愠,不亦君子乎?.</p>
<small>出自<cite title = "Source Title">《论语》</cite></small>
</blockquote>
</div>
<div class = "col-lg-3">

<h2>无序列表</h2>
<ul>
<li>水果</li>
<li>蔬菜</li>
<ul>
<li>白菜</li>
<li>青菜</li>
<li>花椰菜</li>
</ul>
<li>肉类</li>
</ul>
```

```
<h2>有序列表</h2>
<ol>
<li>苹果</li>
<li>橙子</li>
<li>香蕉</li>
</ol>
<h2>无样式列表</h2>
<ul class = "list-unstyled">
<li>苹果</li>
<li>橙子</li>
<li>香蕉</li>
</ul>
<h2>内联列表</h2>
<ul class = "list-inline">
<li>苹果</li>
<li>橙子</li>
<li>香蕉</li>
</ul>
</div>
<div class = "col-lg-3">
<h2>描述性列表</h2>
<dl>
<dt>苹果</dt>
<dd>苹果是一种富含维生素 C 的水果</dd>
<dt>香蕉</dt>
<dd>香蕉中的营养非常丰富</dd>
</dl>
<dl class = "dl-horizontal">
<h2>水平描述性列表</h2>
<dt>梨</dt>
<dd>秋天干燥,适合吃梨,润肺</dd>
<dt>橙子</dt>
<dd>橙子是一种适合冬天吃的水果</dd>
</dl>
</div>
</div>
</div><! -- /.row -->
```

8.4.2　表　格

在表格组件里,Bootstrap 提供了 1 种基础. table 样式、4 种附加样式(. table - striped、. table - bordered、. table - hover、. table - condensed)以及一个支持响应式布局的. table - responsive 容器样式,每种附加特效都是在. table 样式的基础上联合应用才生效的。

Bootstrap 主要提供了 4 个 css 的类来控制表格的边和结构。表 8 - 1 列出了 bootstrap 的

table 可选项。

表 8-1　表格选项(Table Options)

名　字	Class	描　述
Default	None	没有样式,只有行和列
Basic	. table	只有在行间有竖线
Bordered	. table - bordered	圆角和添加外边框
Zebra - stripe	. table - striped	为奇数行添加淡灰色的背景色
Condensed	. table - condensed	将横向的 padding 对切

图 8-12 所示为实现的不带边框的表格效果图。实现的代码片段如下。

不带边框的表格

姓名	学号	12月1日	12月2日
张三	126599	到课	旷课
李四	124399	到课	到课
王五	124499	到课	到课
赵六	126449	旷课	到课

不带边框、奇数行带背景的表格

姓名	学号	12月1日	12月2日
张三	126599	到课	旷课
李四	124399	到课	到课
王五	124499	到课	到课
赵六	126449	旷课	到课

图 8-12　不带边框的表格效果

```
<div class = "col-lg-6">
<h2 class = "page-header" >不带边框的表格</h2>
<div class = "table-responsive">
<table class = "table  table-hover tablesorter">
<thead>
<tr>
<th>姓名</th>
<th>学号</th>
<th>12 月 1 日</th>
<th>12 月 2 日</th>
</tr>
</thead>
<tbody>
<tr>
<td>张三</td>
<td>126599</td>
<td>到课</td>
<td>旷课</td>
</tr>
<tr>
<td>李四</td>
<td>124399</td>
<td>到课</td>
```

```
<td>到课</td>
</tr>
<tr>
<td>王五</td>
<td>124499</td>
<td>到课</td>
<td>到课</td>
</tr>
<tr>
<td>赵六</td>
<td>126449</td>
<td>旷课</td>
<td>到课</td>
</tr>
</tbody>
</table>
</div>
</div>
```

如果将上述代码中的<table class="table　table - hover tablesorter">加入 table - striped 将出现奇数行带淡灰色背景，如果加入 table - bordered，将出现边框，效果如图 8 - 13 所示。

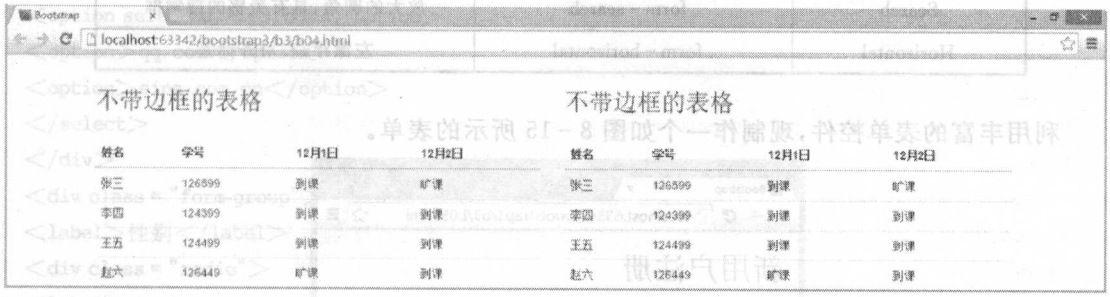


图 8 - 13　带边框的表格效果

在 tr 标记上，加入 class="active"或者 class="success"或者 class="danger"，如下代码片段所示。

```
<tr class = "active">
    <td>张三</td>
    <td>126599</td>
    <td>到课</td>
    <td>旷课</td>
</tr>
```

则该行将出现带有成功绿、警示黄、危险红等提示效果的背景，如图 8 - 14 所示。

有提示背景的表格

姓名	学号	12月1日	12月2日
张三	126599	到课	旷课
李四	124399	到课	到课
王五	124499	到课	到课
赵六	126449	旷课	到课

图 8-14　带提示背景的表格效果

8.4.3　表　单

表单(Form)是 HTML 网页交互中最重要的部分，Bootstrap 框架为表单提供了丰富的样式(基础、内联、横向)。结合丰富多样的表单控件，可以组合出绚丽多彩的表单。BootStrap 主要提供了四种表单选项，如表 8-2 所示。

表 8-2　表单选项

名　字	Class	描　述
Vertical (default)	.form-vertical (not required)	堆放式，可控制的左对齐标签
Inline	.form-inline	左对齐标签和简约的内联控制块
Search	.form-search	放大的圆角，具有美感的搜索框
Horizontal	.form-horizontal	左漂浮，右对齐标签

利用丰富的表单控件，现制作一个如图 8-15 所示的表单。

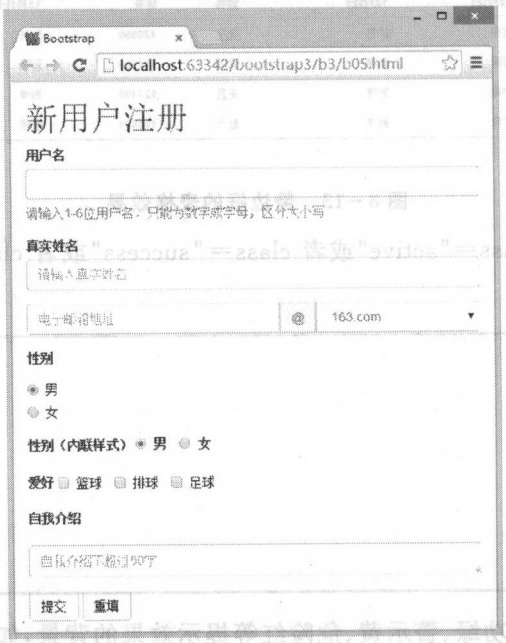


图 8-15　用户注册表单

实现的代码片段如下。

```
<body>
<div class = "container">
<h1>新用户注册</h1>
<div class = "row">
<div class = "col-lg-6">
<form role = "form">
<div class = "form-group">
<label>用户名</label>
<input class = "form-control"/>
<p class = "help-block">请输入 1-6 位用户名,只能为数字或字母,区分大小写</p>
</div>
<div class = "form-group">
<label>真实姓名</label>
<input class = "form-control" placeholder = "请输入真实姓名"/>
</div>
<div class = "form-group input-group">

<input class = "form-control" placeholder = "电子邮箱地址"/>
<span class = "input-group-addon">@</span>
<select class = "form-control">
<option selected>163.com</option>
<option>qq.com</option>
<option>sina.com.cn</option>
</select>
</div>
<div class = "form-group">
<label>性别</label>
<div class = "radio">
<label>
<input type = "radio" name = "optionsRadios" id = "optionsRadios1" value = "option1" checked>
男
</label>
</div>
<div class = "radio">
<label>
<input type = "radio" name = "optionsRadios" id = "optionsRadios2" value = "option2">
女
</label>
</div>
</div>
<div class = "form-group">
<label>性别(内联样式)</label>
<div class = "radio-inline">
```

```
<label>
<input type = "radio" name = "optionsRadios1" id = "optionsRadios3" value = "option1" checked>
男
</label>
</div>
<div class = "radio-inline">
<label>
<input type = "radio" name = "optionsRadios1" id = "optionsRadios4" value = "option2">
女
</label>
</div>
</div>
<div class = "form-group">
<label>爱好</label>
<label class = "checkbox-inline">
<input type = "checkbox">篮球
</label>
<label class = "checkbox-inline">
<input type = "checkbox">排球
</label>
<label class = "checkbox-inline">
<input type = "checkbox">足球
</label>
</div>
<div class = "form-group">
<label>自我介绍</label>
</div>
<div class = "form-group">
<textarea class = "form-control " rows = "1" placeholder = "自我介绍不超过 50 字">
</textarea>
</div>
<button type = "submit" class = "btn btn-default">提交</button>
<button type = "reset" class = "btn btn-default">重填</button>
</form>
</div>
</div>
</div>
</body>
```

上述代码中，label 和 input 元素放在一个样式为. form - group 的 div 里，. form - group 样式提供了一个 margin - bottom:15px 的底部外边距，所以可以很清晰地看到每一组控件。

在 select、input、textarea 元素上应用了. form - control 样式，显示的宽度会变成 100%，并且 placeholder 的颜色都设置成了 #999999，此时更改 bootstrap. css 中的对应代码，可以获得自定义的风格样式。

在 textarea 元素里定义了 rows 数字即可定义文本框的高度，定义 cols 可以定义大文本框的宽度。由于. form - control 样式的表单控件都设置了 100%的宽度(或 auto)。设置了该样式. form - control，就不需要再设置 cols 属性。

checkbox 和 radio 通常在使用的过程中和 label 文字搭配，但通常会出现左右边距无法对齐的问题。为此，Bootstrap 进行了标准设置，即使用的时候，每个 input 外部都要用 label 包住，并且在最外层用容器元素包住，并应用相应的. checkbox 和. radio 样式。

8.4.4　按　钮

Bootstrap 提供多种样式的按钮，同样是通过 CSS 的类来控制，包括 btn，btn - primary，btn - info，btn - success 等不同颜色的按钮，亦可以简单通过. btn - large、. btn - mini 等 CSS 的 class 控制按钮大小，能够同时用在<a>，<button>，<input>标签上，非常简单易用。按钮基本效果图如 8 - 16 所示。

图 8 - 16　按钮基本效果图

实现的代码片段如下。

```
<body>
<div class = "container">
<div class = "row">
<div class = "col-lg-8">
<h2>按钮效果</h2>
<table class = "table table-bordered table-striped">
<thead>
<tr>
```

```
<th>Button</th>
<th>class = ""</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><button class = "btn" href = "#">Default</button></td>
<td><code>btn</code></td>
<td>Standard gray button with gradient</td>
</tr>
<tr>
<td><button class = "btn btn-primary" href = "#">Primary</button></td>
<td><code>btn btn-primary</code></td>
<td>Provides extra visual weight and identifies the primary action in a set of buttons</td>
</tr>
<tr>
<td><button class = "btn btn-info" href = "#">Info</button></td>
<td><code>btn btn-info</code></td>
<td>Used as an alternative to the default styles</td>
</tr>
<tr>
<td><button class = "btn btn-success" href = "#">Success</button></td>
<td><code>btn btn-success</code></td>
<td>Indicates a successful or positive action</td>
</tr>
<tr>
<td><button class = "btn btn-warning" href = "#">Warning</button></td>
<td><code>btn btn-warning</code></td>
<td>Indicates caution should be taken with this action</td>
</tr>
<tr>
<td><button class = "btn btn-danger" href = "#">Danger</button></td>
<td><code>btn btn-danger</code></td>
<td>Indicates a dangerous or potentially negative action</td>
</tr>
<tr>
<td><button class = "btn btn-inverse" href = "#">Inverse</button></td>
<td><code>btn btn-inverse</code></td>
<td>Alternate dark gray button, not tied to a semantic action or use</td>
</tr>
</tbody>
</table>
</div>
</div>
```

```
</div>
</body>
```

像 input 一样，Bootstrap 也提供了控制 button 按钮大小的 CSS 样式，它们是：.btn-lg、.btn-sm、.btn-xs。这些样式可以和 btn 颜色样式组合使用。特别要注意的是，在使用下拉菜单前，需要引入所需要的 JS 文件，包括 Bootstrap 自带 JS 和 Jquery 1.9.0 以上版本的 JS 文件（后文中提及的带有响应式的组件，也同样需要），并保证 Jquery 的 JS 文件的引入。示例如图 8-17 所示。

图 8-17　按钮大小、风格设置图

实现的代码片段如下。

引入的 CSS 和 JS 文件代码为：

```
<link href = "../bootstrap/css/bootstrap.min.css" rel = "stylesheet" media = "screen">
<link href = "../bootstrap/css/bootstrap-theme.css" rel = "stylesheet" media = "screen">
<script src = "../bootstrap/js/jquery-2.1.1.js"></script>
<script src = "../bootstrap/js/bootstrap.min.js"></script>
```

按钮效果代码为：

```
<div class = "row">
<div class = "col-lg-12">
<h1 id = "buttons" class = "page-header">按钮大小、风格设置效果</h1>
</div>
</div><!-- /.row -->

<div class = "row">
```

```
<div class = "col-lg-6">
    <h2>基本按钮样式</h2>
<p>
<button type = "button" class = "btn btn-default">Default</button>
<button type = "button" class = "btn btn-primary">Primary</button>
<button type = "button" class = "btn btn-success">Success</button>
<button type = "button" class = "btn btn-info">Info</button>
<button type = "button" class = "btn btn-warning">Warning</button>
<button type = "button" class = "btn btn-danger">Danger</button>
<button type = "button" class = "btn btn-link">Link</button>
</p>
<h2>不可用按钮样式</h2>
<p>
<button type = "button" class = "btn btn-default disabled">Default</button>
<button type = "button" class = "btn btn-primary disabled">Primary</button>
<button type = "button" class = "btn btn-success disabled">Success</button>
<button type = "button" class = "btn btn-info disabled">Info</button>
<button type = "button" class = "btn btn-warning disabled">Warning</button>
<button type = "button" class = "btn btn-danger disabled">Danger</button>
<button type = "button" class = "btn btn-link disabled">Link</button>
</p>
<h2>下拉菜单按钮样式</h2>
<p>
<div class = "btn-group">
<button type = "button" class = "btn btn-default">Default</button>
<button type = "button" class = "btn btn-default dropdown-toggle" data-toggle = "dropdown">
<span class = "caret"></span>
</button>
<ul class = "dropdown-menu">
<li><a href = "#">Action</a></li>
<li><a href = "#">Another action</a></li>
<li><a href = "#">Something else here</a></li>
<li class = "divider"></li>
<li><a href = "#">Separated link</a></li>
</ul>
</div><! -- /btn-group -->
<div class = "btn-group">
<button type = "button" class = "btn btn-primary">Primary</button>
<button type = "button" class = "btn btn-primary dropdown-toggle" data-toggle = "dropdown">
<span class = "caret"></span>
</button>
<ul class = "dropdown-menu">
<li><a href = "#">Action</a></li>
```

```
<li><a href = "#">Another action</a></li>
<li><a href = "#">Something else here</a></li>
<li class = "divider"></li>
<li><a href = "#">Separated link</a></li>
</ul>
</div><! -- /btn-group -->
<div class = "btn-group">
<button type = "button" class = "btn btn-success">Success</button>
<button type = "button" class = "btn btn-success dropdown-toggle" data-toggle = "dropdown">
<span class = "caret"></span>
</button>
<ul class = "dropdown-menu">
<li><a href = "#">Action</a></li>
<li><a href = "#">Another action</a></li>
<li><a href = "#">Something else here</a></li>
<li class = "divider"></li>
<li><a href = "#">Separated link</a></li>
</ul>
</div><! -- /btn-group -->
<div class = "btn-group">
<button type = "button" class = "btn btn-info">Info</button>
<button type = "button" class = "btn btn-info dropdown-toggle" data-toggle = "dropdown">
<span class = "caret"></span>
</button>
<ul class = "dropdown-menu">
<li><a href = "#">Action</a></li>
<li><a href = "#">Another action</a></li>
<li><a href = "#">Something else here</a></li>
<li class = "divider"></li>
<li><a href = "#">Separated link</a></li>
</ul>
</div><! -- /btn-group -->
<div class = "btn-group">
<button type = "button" class = "btn btn-warning">Warning</button>
<button type = "button" class = "btn btn-warning dropdown-toggle" data-toggle = "dropdown">
<span class = "caret"></span>
</button>
<ul class = "dropdown-menu">
<li><a href = "#">Action</a></li>
<li><a href = "#">Another action</a></li>
<li><a href = "#">Something else here</a></li>
<li class = "divider"></li>
<li><a href = "#">Separated link</a></li>
```

```
</ul>
</div><! -- /btn-group -->
</p>
<h2>不同大小的按钮样式</h2>
<p>
<button type = "button" class = "btn btn-primary btn-lg">Large button</button>
<button type = "button" class = "btn btn-primary">Default button</button>
<button type = "button" class = "btn btn-primary btn-sm">Small button</button>
<button type = "button" class = "btn btn-primary btn-xs">Mini button
</button>
</p>

</div>

<div class = "col-lg-6">
<h2>块状按钮样式</h2>
<p>
<button type = "button" class = "btn btn-default btn-lg btn-block">
Block level button
</button>
</p>

<p>
<div class = "btn-group btn-group-justified">
<a href = "#" class = "btn btn-default">Left</a>
<a href = "#" class = "btn btn-default">Right</a>
<a href = "#" class = "btn btn-default">Middle</a>
</div>
</p>
<h2>水平按钮组样式</h2>
<p>
<div class = "btn-toolbar">
<div class = "btn-group">
<button type = "button" class = "btn btn-default">1</button>
<button type = "button" class = "btn btn-default">2</button>
<button type = "button" class = "btn btn-default">3</button>
<button type = "button" class = "btn btn-default">4</button>
</div>
<div class = "btn-group">
<button type = "button" class = "btn btn-default">5</button>
<button type = "button" class = "btn btn-default">6</button>
<button type = "button" class = "btn btn-default">7</button>
</div>
```

```
<div class = "btn-group">
<button type = "button" class = "btn btn-default">8</button>

<div class = "btn-group">
<button type = "button" class = "btn btn-default dropdown-toggle" data-toggle = "dropdown">
Dropdown
<span class = "caret"></span>
</button>
<ul class = "dropdown-menu">
<li><a href = "#">Dropdown link</a></li>
<li><a href = "#">Dropdown link</a></li>
<li><a href = "#">Dropdown link</a></li>
</ul>
</div>
</div>
</div>
</p>
<h2>垂直按钮组样式</h2>
<p>
<div class = "btn-group-vertical">
<button type = "button" class = "btn btn-default">Button</button>
<button type = "button" class = "btn btn-default">Button</button>
<button type = "button" class = "btn btn-default">Button</button>
<button type = "button" class = "btn btn-default">Button</button>
</div>
</p>

</div>
</div><! -- /.row -->
```

8.5 BootStrap 组件

前面三个小节讲授的是 BootStrap 的最基本功能，如果想实现更丰富的页面效果，还需要使用 BootStrap 组件。Bootstrap 内置几十种高效可用的组件，以实现导航栏、通知、弹出框等功能。

8.5.1 导航(navigation)

在 BootStrap 中，所有的导航组件，都必须使用. nav 的类实现基础的导航标签。Bootstrap 提供了灵活多变的导航实现方案，允许使用同样的标签，不同的 CSS 类，定制出不同样式的导航条。

其中，基本标签导航使用 CSS 类 nav nav - tabs；基本胶囊式标签导航使用 nav nav - pills；此外，还可以将导航进行竖排，使用 CSS 类 nav nav - tabs nav - stacked 和 nav nav - pills nav - stacked。实现这几项导航的效果图如图 8 - 18 所示。

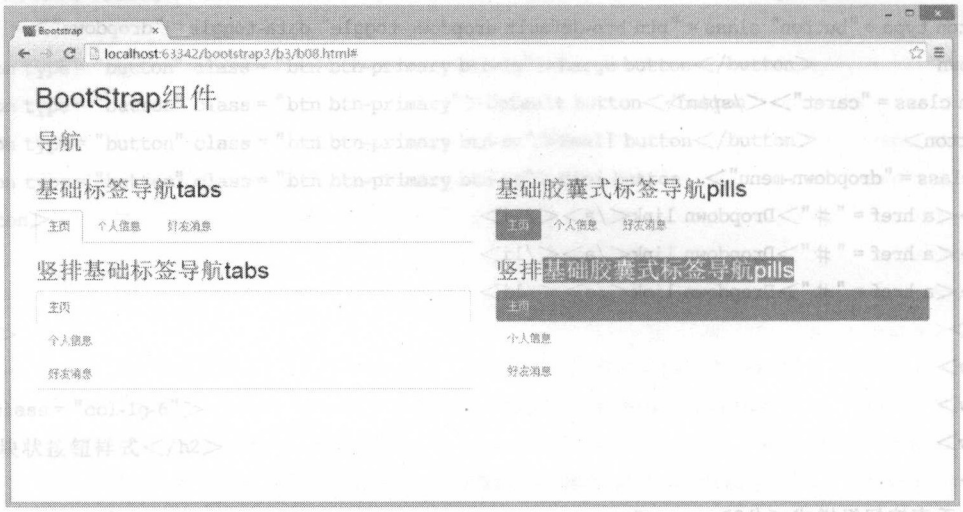


图 8 - 18　不同风格的基础导航

实现的代码片段如下。

```
<div class = "container">
    <h1>BootStrap 组件</h1>
    <h2>导航</h2>
    <div class = "row">
    <div class = "col-lg-6">
    <h2>基础标签导航 tabs</h2>
<ul class = "nav nav-tabs">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">个人信息</a></li>
<li><a href = "#">好友消息</a></li>
</ul>
</div>
<div class = "col-lg-6">
    <h2>基础胶囊式标签导航 pills</h2>
    <ul class = "nav nav-pills">
    <li class = "active"><a href = "#">主页</a></li>
    <li><a href = "#">个人信息</a></li>
    <li><a href = "#">好友消息</a></li>
    </ul>
    </div>
    </div>
    <div class = "row">
    <div class = "col-lg-6">
```

```
    <h2>竖排基础标签导航 tabs</h2>
<ul class = "nav nav-tabs nav-stacked">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">个人信息</a></li>
<li><a href = "#">好友消息</a></li>
</ul>
</div>
    <div class = "col-lg-6">
    <h2>竖排基础胶囊式标签导航 pills</h2>
    <ul class = "nav nav-pills nav-stacked">
    <li class = "active"><a href = "#">主页</a></li>
    <li><a href = "#">个人信息</a></li>
    <li><a href = "#">好友消息</a></li>
</ul>
</div>
</div>
</div>
```

利用 BootStrap 提供的 CSS 类，在基础标签导航的基础上，还可以使用下拉菜单式导航和列表式的导航，其用法与基础标签导航类似，也十分简单易用。CSS 类 nav nav - list 用于实现列表式导航，CSS 类 dropdown - menu 用于实现下拉菜单式导航。实现效果如图 8 - 19 所示。

图 8 - 19　下拉菜单式与列表式导航

实现的代码片段如下。

```
<div class = "row">
<div class = "col - lg - 6">
<h2>基础列表导航 Nav List</h2>
```

```
<ul class = "nav nav - list">
<li class = "nav - header">个人管理</li>
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">个人信息</a></li>
<li><a href = "#">修改密码</a></li>
<li class = "nav - header">好友圈</li>
<li><a href = "#">好友消息</a></li>
<li><a href = "#">好友设置</a></li>
<li class = "divider"></li>
<li><a href = "#">帮助</a></li>
</ul>
</div>
<div class = "col - lg - 6">
<h2>图标列表导航</h2>
<ul class = "nav nav - list">
<li class = "nav - header">个人管理</li>
<li class = "active"><a href = "#"><i class = "glyphicon glyphicon - white glyphicon - home">
</i>主页</a>
</li>
<li><a href = "#"><i class = "glyphicon glyphicon - book"></i>个人信息</a></li>
<li><a href = "#"><i class = "glyphicon glyphicon - pencil"></i>修改密码</a></li>
<li class = "nav - header">好友圈</li>
<li><a href = "#"><i class = "glyphicon glyphicon - user"></i>好友消息</a></li>
<li><a href = "#"><i class = "glyphicon glyphicon - cog"></i>好友设置</a></li>
<li class = "divider"></li>
<li><a href = "#"><i class = "glyphicon glyphicon - flag"></i>帮助</a></li>
</ul>
</div>
</div>
<div class = "row">
<div class = "col - lg - 6">
<h2>pills 式的下拉菜单导航</h2>
<ul class = "nav nav - pills">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">帮助</a></li>
<li class = "dropdown">
<a id = "drop4" role = "button" data - toggle = "dropdown" href = "#">个人管理<b class = "caret">
</b>
</a>
<ul id = "menu2" class = "dropdown - menu" role = "menu" aria - labelledby = "drop4">
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
修改密码</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
个人信息</a>
```

```
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
好友消息</a>
</li>
<li role = "presentation" class = "divider">好友管理</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
好友设置</a>
</li>
</ul>
</li>
</ul>
</div>
<div class = "col - lg - 6">
<h2>tab 式的下拉菜单导航</h2>
<ul class = "nav nav - tabs">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">帮助</a></li>
<li class = "dropdown">
<a id = "drop5" role = "button" data - toggle = "dropdown" href = "#">个人管理<b class = "caret">
</b>
</a>
<ul id = "menu3" class = "dropdown - menu" role = "menu" aria - labelledby = "drop5">
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
修改密码</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
个人信息</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
好友消息</a>
</li>
<li role = "presentation" class = "divider">好友管理</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
好友设置</a>
</li>
</ul>
</li>
</ul>
</div>
</div>
<div class = "row">
<div class = "col - lg - 6">
<h2>基础列表导航 Nav List</h2>
<ul class = "nav nav - list">
<li class = "nav - header">个人管理</li>
```

```
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">个人信息</a></li>
<li><a href = "#">修改密码</a></li>
<li class = "nav - header">好友圈</li>
<li><a href = "#">好友消息</a></li>
<li><a href = "#">好友设置</a></li>
<li class = "divider"></li>
<li><a href = "#">帮助</a></li>
</ul>
</div>
<div class = "col - lg - 6">
<h2>图标列表导航</h2>
<ul class = "nav nav - list">
<li class = "nav - header">个人管理</li>
<li class = "active"><a href = "#"><i class = "glyphicon glyphicon - white glyphicon - home">
</i>主页</a>
</li>
<li><a href = "#"><i class = "glyphicon glyphicon - book"></i>个人信息</a></li>
<li><a href = "#"><i class = "glyphicon glyphicon - pencil"></i>修改密码</a></li>
<li class = "nav - header">好友圈</li>
<li><a href = "#"><i class = "glyphicon glyphicon - user"></i>好友消息</a></li>
<li><a href = "#"><i class = "glyphicon glyphicon - cog"></i>好友设置</a></li>
<li class = "divider"></li>
<li><a href = "#"><i class = "glyphicon glyphicon - flag"></i>帮助</a></li>
</ul>
</div>
</div>
<div class = "row">
<div class = "col - lg - 6">
<h2>pills 式的下拉菜单导航</h2>
<ul class = "nav nav - pills">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">帮助</a></li>
<li class = "dropdown">
<a id = "drop4" role = "button" data - toggle = "dropdown" href = "#">个人管理<b class = "caret">
</b>
</a>
<ul id = "menu2" class = "dropdown - menu" role = "menu" aria - labelledby = "drop4">
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
修改密码</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
个人信息</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
```

```
好友消息</a>
</li>
<li role = "presentation" class = "divider">好友管理</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
好友设置</a>
</li>
</ul>
</li>
</ul>
</div>
<div class = "col - lg - 6">
<h2>tab 式的下拉菜单导航</h2>
<ul class = "nav nav - tabs">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">帮助</a></li>
<li class = "dropdown">
<a id = "drop5" role = "button" data - toggle = "dropdown" href = "#">个人管理<b class = "caret">
</b>
</a>
<ul id = "menu3" class = "dropdown - menu" role = "menu" aria - labelledby = "drop5">
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
修改密码</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
个人信息</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
好友消息</a>
</li>
<li role = "presentation" class = "divider">好友管理</li>
<li role = "presentation"><a role = "menuitem" tabindex = " - 1" href = "http://twitter.com/fat">
好友设置</a>
</li>
</ul>
</li>
</ul>
</div>
</div>
```

8.5.2　导航条(navbar)

导航条是网页设计中不可缺少的部分,其目的是让网站的层次结构以一种有条理的方式清晰展示,并引导用户毫不费力地找到并管理信息,让用户在浏览网站过程中不至迷失。因此,导航图是页面中非常重要的组成部分。BootStrap 提供导航条组件,使得大家能够轻松的

实现当前流行的简洁导航风格。

在 BootStrap 中，导航条的基础类不再是 nav 而是 navbar，默认的导航图是在页面的顶部，并不固定。将导航条放在任意一个 container 中，其宽度与页面内容同宽。

在最外层 div 上应用，.navbar 添加一个额外的属性 navbar - fixed - top 与 navbar - fixed - bottom，就可以将导航条固定到顶部或是底部。另外，还可通过 navbar navbar - inverse 将导航条进行反色显示。此外，在导航条中运用 8.5.6 节中的导航等知识，可以创建不同的导航项，在导航项中引入表单，即可获得简易搜索条功能，这一切只需要少量代码即可实现，更显出 BootStrap 的易用性。接下来实现如图 8 - 20 的效果图。

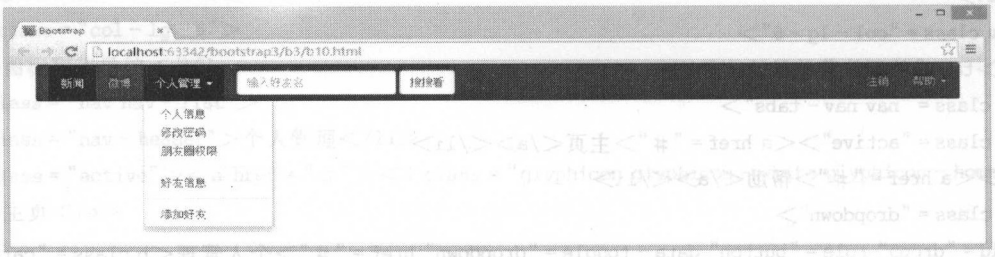


图 8 - 20　基本导航条

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-12">
<div class = "navbar navbar-inverse">
<div class = "collapse navbar-collapse" id = "bs-example-navbar-collapse-1">
<ul class = "nav navbar-nav">
<li class = "active"><a href = "#">新闻</a></li>
<li><a href = "#">微博</a></li>
<li class = "dropdown">
<a href = "#" class = "dropdown-toggle" data-toggle = "dropdown">个人管理<b class = "caret"></b>
</a>
<ul class = "dropdown-menu">
<li><a href = "#">个人信息</a></li>
<li><a href = "#">修改密码</a></li>
<li><a href = "#">朋友圈权限</a></li>
<li class = "divider"></li>
<li><a href = "#">好友信息</a></li>
<li class = "divider"></li>
<li><a href = "#">添加好友</a></li>
</ul>
</li>
</ul>
<form class = "navbar-form navbar-left" role = "search">
<div class = "form-group">
<input type = "text" class = "form-control" placeholder = "输入好友名">
```

```
</div>
<button type = "submit" class = "btn btn-default">搜搜看</button>
</form>
<ul class = "nav navbar-nav navbar-right">
<li><a href = "#">注销</a></li>
<li class = "dropdown">
<a href = "#" class = "dropdown-toggle" data-toggle = "dropdown">帮助<b class = "caret"></b>
</a>
<ul class = "dropdown-menu">
<li><a href = "#">使用教程</a></li>
<li><a href = "#">反馈意见</a></li>
</ul>
</li>
</ul>
</div>
</div>
</div>
</div><div class = "row">
<div class = "col-lg-6">
<h2>基础列表导航 Nav List</h2>
<ul class = "nav nav-list">
<li class = "nav-header">个人管理</li>
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">个人信息</a></li>
<li><a href = "#">修改密码</a></li>
<li class = "nav-header">好友圈</li>
<li><a href = "#">好友消息</a></li>
<li><a href = "#">好友设置</a></li>
<li class = "divider"></li>
<li><a href = "#">帮助</a></li>
</ul>
</div>
<div class = "col-lg-6">
<h2>图标列表导航</h2>
<ul class = "nav nav-list">
<li class = "nav-header">个人管理</li>
<li class = "active"><a href = "#"><i class = "glyphicon glyphicon-white glyphicon-home">
</i>主页</a>
</li>
<li><a href = "#"><i class = "glyphicon glyphicon-book"></i>个人信息</a></li>
<li><a href = "#"><i class = "glyphicon glyphicon-pencil"></i>修改密码</a></li>
<li class = "nav-header">好友圈</li>
<li><a href = "#"><i class = "glyphicon glyphicon-user"></i>好友消息</a></li>
<li><a href = "#"><i class = "glyphicon glyphicon-cog"></i>好友设置</a></li>
<li class = "divider"></li>
```

```
<li><a href = "#"><i class = "glyphicon glyphicon-flag"></i>帮助</a></li>
</ul>
</div>
</div>

<div class = "row">
<div class = "col-lg-6">
<h2>pills 式的下拉菜单导航</h2>
<ul class = "nav nav-pills">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">帮助</a></li>
<li class = "dropdown">
<a id = "drop4" role = "button" data-toggle = "dropdown" href = "#">个人管理<b class = "caret">
</b>
</a>
<ul id = "menu2" class = "dropdown-menu" role = "menu" aria-labelledby = "drop4">
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">修
改密码</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">个
人信息</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">好
友消息</a>
</li>
<li role = "presentation" class = "divider">好友管理</li>
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">好
友设置</a>
</li>
</ul>
</li>
</ul>
</div>
<div class = "col-lg-6">
<h2>tab 式的下拉菜单导航</h2>
<ul class = "nav nav-tabs">
<li class = "active"><a href = "#">主页</a></li>
<li><a href = "#">帮助</a></li>
<li class = "dropdown">
<a id = "drop5" role = "button" data-toggle = "dropdown" href = "#">个人管理<b class = "caret">
</b>
</a>
<ul id = "menu3" class = "dropdown-menu" role = "menu" aria-labelledby = "drop5">
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">修
改密码</a>
```

```
</li>
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">个人信息</a>
</li>
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">好友消息</a>
</li>
<li role = "presentation" class = "divider">好友管理</li>
<li role = "presentation"><a role = "menuitem" tabindex = "-1" href = "http://twitter.com/fat">好友设置</a>
</li>
</ul>
</li>
</ul>
</div>
</div>
```

面包屑(breadcrumbs)导航用作显示用户在网站或者 App 的位置,让使用者获得更好的用户体验。实现后效果如图 8-21 所示。

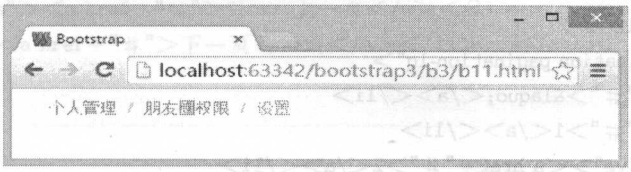


图 8-21　面包屑导航条

实现的代码片段如下。

```
<div class = "row">
    <div class = "col-lg-12">
    <ol class = "breadcrumb">
    <li><a href = "#">个人管理</a></li>
    <li><a href = "#">朋友圈权限</a></li>
    <li class = "active">设置</li>
    </ol>
    </div>
</div>
```

页码(pagination)也是非常常用的页面要素,Bootstrap 提供两种风格的翻页组件。一个是多页面导航,用于多个页码的跳转;另一种则是 Pager,是轻量级组件,可以快速翻动上下页,适用于个人博客或者杂志。

在 BootStrap 中,使用 CSS 类 pagination 实现分页导航,并可以使用 CSS 类 active 来确定当前页,使用 CSS 类 pagination-lg 和 pagination-sm 来控制分页导航的大小,使用 CSS 类

pager 实现简洁版分页显示。图 8-22 为实现的各种分页导航条效果图。

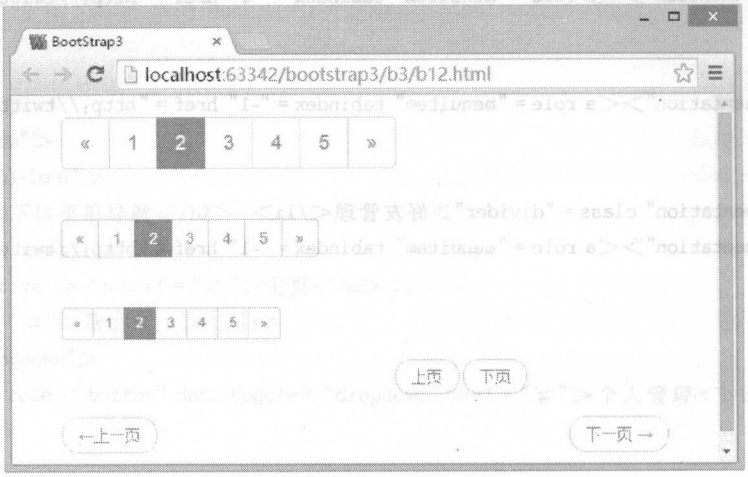


图 8-22　分页导航条

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-12">
    <ul class = "pagination pagination-lg">
    <li><a href = "#">&laquo;</a></li>
    <li><a href = "#">1</a></li>
    <li class = "active"><a href = "#">2</a></li>
    <li><a href = "#">3</a></li>
    <li><a href = "#">4</a></li>
    <li><a href = "#">5</a></li>
    <li><a href = "#">&raquo;</a></li>
    </ul>
    </div>
    </div>
    <div class = "row">
    <div class = "col-lg-12">

<ul class = "pagination">
<li><a href = "#">&laquo;</a></li>
<li><a href = "#">1</a></li>
<li class = "active"><a href = "#">2</a></li>
<li><a href = "#">3</a></li>
<li><a href = "#">4</a></li>
<li><a href = "#">5</a></li>
<li><a href = "#">&raquo;</a></li>
</ul>
</div>
</div>
```

```
<div class = "row">
<div class = "col-lg-12">
<ul class = "pagination pagination-sm">
<li><a href = "#">&laquo;</a></li>
<li><a href = "#">1</a></li>
<li class = "active"><a href = "#">2</a></li>
<li><a href = "#">3</a></li>
<li><a href = "#">4</a></li>
<li><a href = "#">5</a></li>
<li><a href = "#">&raquo;</a></li>
</ul>
</div>
</div>
<div class = "row">
<div class = "col-lg-6">
<ul class = "pager">
<li><a href = "#">上页</a></li>
<li><a href = "#">下页</a></li>
</ul>
<ul class = "pager">
<li class = "previous"><a href = "#">&larr;上一页</a></li>
<li class = "next"><a href = "#">下一页 &rarr;</a></li>
</ul>
</div>
</div>
```

8.5.3　标签(label)和徽章(badges)

标签可用于计数、提示或页面上其他的标记显示，是 HTML 页面中非常好用的一个元素，通过不同的 CSS 类，标签会显示不同的颜色，起到不同的提示作用。徽章的作用与标签类似。使用徽章，可以给链接、Bootstrap 导航等加入，实现突出显示新的或未读的条目。图 8-23 是用标签实现的不同格式的标题的效果图。

图 8-23　标签效果图

实现的代码片段如下。

```
<div class = "col-lg-8">
    <h1>新闻标题 1<span class = "label label-default">new~</span></h1>
    <h2>新闻标题 2<span class = "label label-success">new~</span></h2>
    <h3>新闻标题 3<span class = "label label-warning">new~</span></h3>
    <h4>新闻标题 4<span class = "label label-danger">new~</span></h4>
    <h5>新闻标题 5<span class = "label label-primary">new~</span></h5>
    <h6>新闻标题 6<span class = "label label-info">new~</span></h6>
</div>
```

此外，在胶囊式导航和列表式导航以及按钮中，使用内置的徽章样式，完成后的效果图如图 8-24 所示。

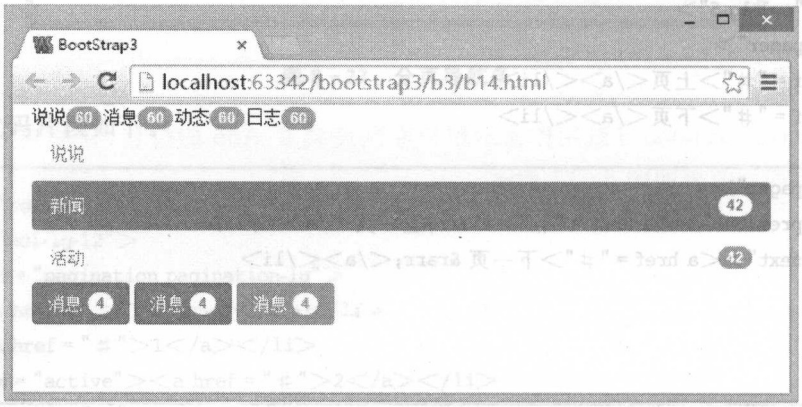


图 8-24　徽章效果图

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-6">
<ul class = "nav nav-pills">
<li>说说<span class = "badge">60</span></li>
<li class = "active">消息<span class = "badge">60</span></li>
<li>动态<span class = "badge">60</span></li>
<li>日志<span class = "badge">60</span></li>
</ul>
<ul class = "nav nav-pills nav-stacked">
<li>
<a href = "#">
说说
</a>
</li>
<li class = "active">
```

```
<a href = "#">
<span class = "badge pull-right">42</span>
新闻
</a>
</li>
<li>
<a href = "#">
<span class = "badge pull-right">42</span>
活动
</a>
</li>
</ul>
<button class = "btn btn-primary" type = "button">
消息<span class = "badge">4</span>
</button>
<button class = "btn btn-danger" type = "button">
消息<span class = "badge">4</span>
</button>
<button class = "btn btn-success" type = "button">
消息<span class = "badge">4</span>
</button>
</div>
</div>
```

8.5.4　缩略图(thumbnails)

缩略图可以实现在网格中布局文本、图像、视频等。通常应用于图片、视屏的搜索结果等页面，通过超级链接可以进一步查看详情。缩略图也具有很好的可定制性，可融入文章片段，按钮等标签。结合排版技术，可以混排不同大小的缩略图，从而实现页面的丰富效果。实现默认形式的 bootstrap 缩略图，只需要简单的 CSS 类 thumbnails。

在图像周围添加带有 class . thumbnail 的<a>标签，就会添加 4 个像素的内边距(padding)和一个灰色的边框。当光标悬停在图像上时，会实时显示出图像的轮廓。如图 8 - 25 所示。

图 8 - 25　基本缩略图

实现的代码片段如下。

```
<div class = "row">
<div class = "col-sm-6 col-md-3">
<a href = "#" class = "thumbnail">
<img src = "img/expo.png" alt = "通用的占位符缩略图">
</a>
</div>
<div class = "col-sm-6 col-md-3">
<a href = "#" class = "thumbnail">
<img src = "img/expo.png" alt = "通用的占位符缩略图">
</a>
</div>
<div class = "col-sm-6 col-md-3">
<a href = "#" class = "thumbnail">
<img src = "img/expo.png" alt = "通用的占位符缩略图">
</a>
</div>
<div class = "col-sm-6 col-md-3">
<a href = "#" class = "thumbnail">
<img src = "img/expo.png" alt = "通用的占位符缩略图">
</a>
</div>
</div>
```

后面介绍实现混编效果缩略图。完成效果如图 8-26 所示。

图 8-26　混排缩略图

实现的代码片段如下。

```
<div class = "row">
<div class = "col-sm-6 col-md-3">
<a href = "#" class = "thumbnail">
<img src = "img/expo.png" alt = "通用的占位符缩略图">
</a>
<div class = "caption">
```

```
<h2>BootStrap 案例</h2>
<p>BootStrap 是一个非常好用的 web 框架</p>
<p>
<a href = "#" class = "btn btn-primary" role = "button">
点赞<span class = "badge">50</span>
</a>
<a href = "#" class = "btn btn-default" role = "button">
评差<span class = "badge">2</span>
</a>
</p>
</div>
</div>
<div class = "col-sm-6 col-md-3">
<a href = "#" class = "thumbnail">
<img src = "img/expo.png" alt = "通用的占位符缩略图">
</a>
</div>
<div class = "col-sm-6 col-md-3">
<a href = "#" class = "thumbnail">
<img src = "img/expo.png" alt = "通用的占位符缩略图">
</a>
</div>
</div>
```

8.5.5　警告框(alert)

在 BootStrap 中，为原有的 alter 类，提供了更多的文字效果，可以加强显示，并使用 CSS 类 alert- dismissable 结合关闭按钮实现可关闭警告框的动态效果。图 8 - 27 为 4 种警告框效果。

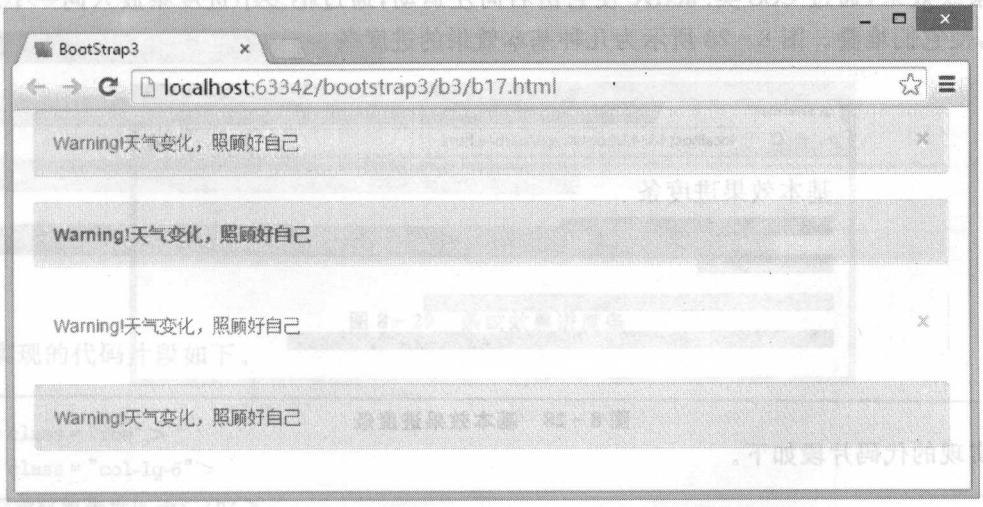


图 8 - 27　警告框效果

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-6">
<div class = "alert alert-success alert-dismissable">
<button type = "button" class = "close" data-dismiss = "alert">&times;
        </button>
        Warning! 天气变化,照顾好自己
</div>
<div class = "alert alert-info"><strong>Warning! 天气变化,照顾好自己</strong>
</div>
    <div class = "alert alert-warning alert-dismissable">
    <button type = "button" class = "close" data-dismiss = "alert">&times;
    </button>
    Warning! 天气变化,照顾好自己
    </div>
<div class = "alert alert-danger">Warning! 天气变化,照顾好自己</div>
</div>
</div>
```

8.5.6 进度条(processing bar)

在页面载入时常常会用到进度条,以便减少用户因等待页面加载而产生的焦虑,提高用户使用感。Bootstrap 提供了多种漂亮、简单、多种颜色的进度条,并结合 CSS3 的渐变(gradients)、透明度(transitions)、动画效果(animations)等特点来实现动态效果。对于 IE7 - 9 和旧版的 Firefox 都不支持这些特性,所以在实现进度条的时候需要注意浏览器的兼容性。

为了一致的样式,进度条使用与按钮和警告框相同的类。通过 CSS 类 progress - striped 实现条纹效果,通过 CSS 类. active 使它由右向左运动,通过把多个进度条放入同一个. progress,使它们堆叠。图 8 - 28 所示为几种基本效果的进度条。

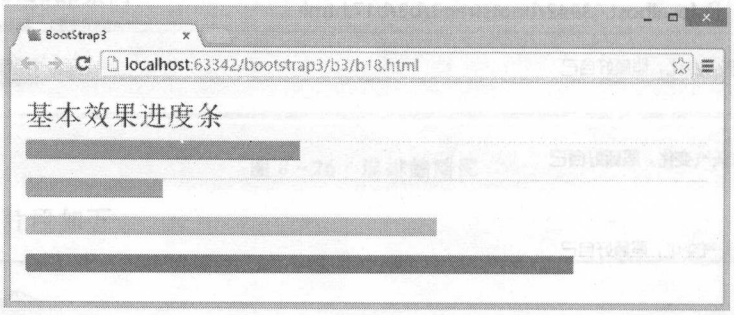


图 8 - 28 基本效果进度条

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-6">
```

```
<h2>基本效果进度条</h2>
<div class = "progress">
<div class = "progress-bar progress-bar-success" role = "progressbar" aria-valuenow = "40"
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 40 % ">
<span class = "sr-only">40 % Complete (success)</span>
</div>
</div>
<div class = "progress">
<div class = "progress-bar progress-bar-info" role = "progressbar" aria-valuenow = "20"
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 20 % ">
<span class = "sr-only">20 % Complete</span>
</div>
</div>
<div class = "progress">
<div class = "progress-bar progress-bar-warning" role = "progressbar" aria-valuenow = "60"
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 60 % ">
<span class = "sr-only">60 % Complete (warning)</span>
</div>
</div>
<div class = "progress">
<div class = "progress-bar progress-bar-danger" role = "progressbar" aria-valuenow = "80"
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 80 % ">
<span class = "sr-only">80 % Complete</span>
</div>
</div>
</div>
```

后面介绍实现几种条纹效果的进度条，完成效果如图 8-29 所示。

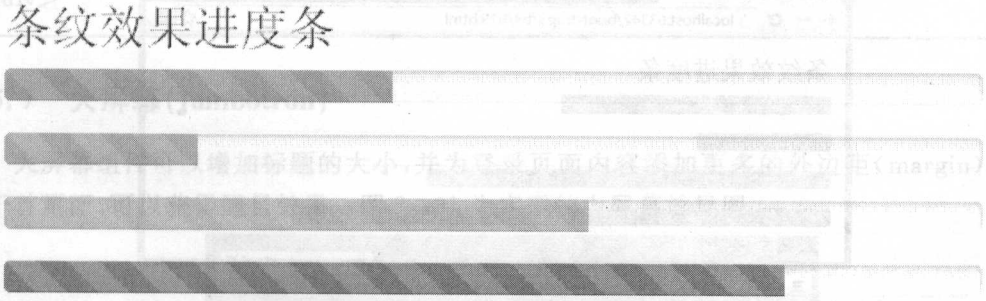


图 8-29　条纹效果进度条

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-6">
<h2>条纹效果进度条</h2>
<div class = "progress progress-striped">
<div class = "progress-bar progress-bar-success" role = "progressbar" aria-valuenow = "40"
```

```
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 40 % ">
<span class = "sr-only">40 % Complete (success)</span>
</div>
</div>
<div class = "progress progress-striped">
<div class = "progress-bar progress-bar-info" role = "progressbar" aria-valuenow = "20"
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 20 % ">
<span class = "sr-only">20 % Complete</span>
</div>
</div>
<div class = "progress progress-striped">
<div class = "progress-bar progress-bar-warning" role = "progressbar" aria-valuenow = "60"
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 60 % ">
<span class = "sr-only">60 % Complete (warning)</span>
</div>
</div>
<div class = "progress progress-striped">
<div class = "progress-bar progress-bar-danger" role = "progressbar" aria-valuenow = "80"
        aria-valuemin = "0" aria-valuemax = "100" style = "width: 80 % ">
<span class = "sr-only">80 % Complete (danger)</span>
</div>
</div>
</div>
</div>
```

最后介绍实现运动效果进度条和堆叠效果进度条，完成效果如图 8-30 所示。

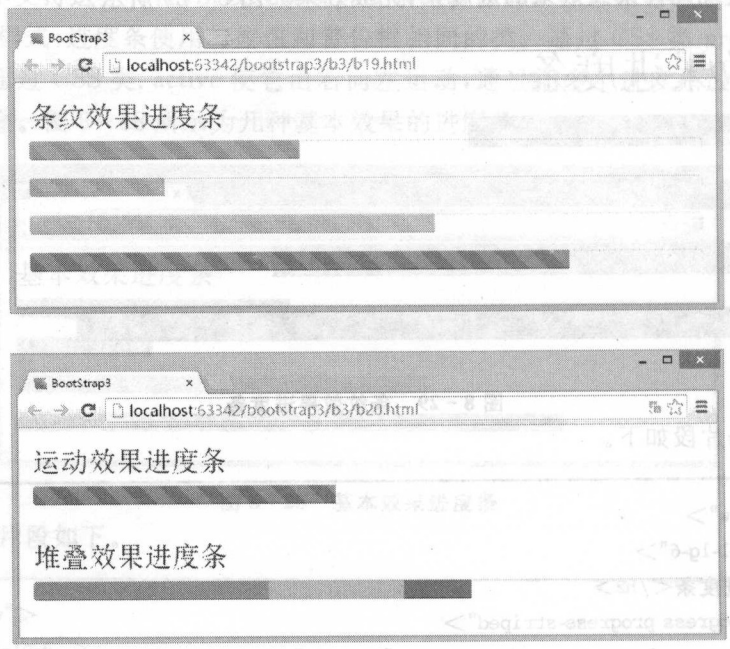


图 8-30　运动与堆叠效果进度条

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-6">
<h2>运动效果进度条</h2>
<div class = "progress progress-striped active">
<div class = "progress-bar"  role = "progressbar" aria-valuenow = "45" aria-valuemin = "0"
      aria-valuemax = "100" style = "width: 45 % ">
<span class = "sr-only">45 % Complete</span>
</div>
</div>
</div>
</div>
<div class = "row">
<div class = "col-lg-6">
<h2>堆叠效果进度条</h2>
<div class = "progress">
<div class = "progress-bar progress-bar-success" style = "width: 35 % ">
<span class = "sr-only">35 % Complete (success)</span>
</div>
<div class = "progress-bar progress-bar-warning" style = "width: 20 % ">
<span class = "sr-only">20 % Complete (warning)</span>
</div>
<div class = "progress-bar progress-bar-danger" style = "width: 10 % ">
<span class = "sr-only">10 % Complete (danger)</span>
</div>
</div>
</div>
</div>
```

8.5.7　大屏幕(jumbotron)

大屏幕组件可以增加标题的大小,并为登录页面内容添加更多的外边距(margin)。通常用于首页面,可以获得醒目效果。图 8-31 为实现的大屏幕效果图。

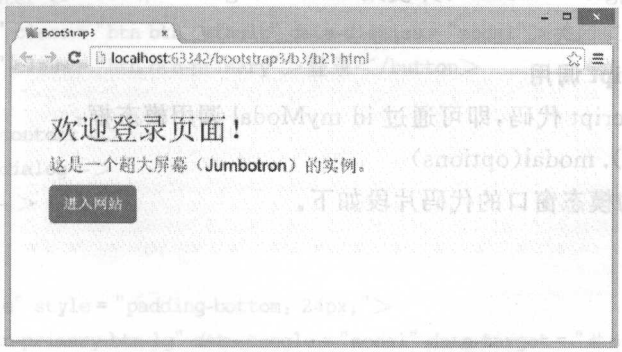


图 8-31　大屏幕效果图

实现的代码片段如下。

```
<div class = "container">
<div class = "jumbotron">
<h1>欢迎登录页面！</h1>
<p>这是一个超大屏幕(Jumbotron)的实例。</p>
<p><a class = "btn btn-primary btn-lg" role = "button">
进入网站</a>
</p>
</div>
</div>
```

8.6 BootStrap 动态效果

BootStrap 作为一套良好的前端工具，要实现现代的动态页面效果，javascript 插件是必不可少的。BootStrap3 提供了 13 个基于 JQuery 类库的插件，包括模态窗口(modals)、滚动监控(scrollspy)、标签效果(tabs)、提示效果(tooltip)、“泡芙”效果(popovers)、警告区域(Alerts)、折叠效果(collapse)、旋转木马(carousel)和输入提示(typeahead)等。bootstrap. js 和 bootstrap. min. js 文件将所有插件包含在一个文件中(前者是未压缩版，后者是压缩版)，在使用 BootStrap 动态效果时，只需要引入这两个 JS 文件中的一个即可。

8.6.1 模态窗口(Modals)

模态窗口是一个可定制响应的轻量级的多功能的 JavaScript 弹出窗口，它可以用来显示弹出式警告、视频、以及网站的图片。模态窗口的源码为 modal. js。需要注意的一点是，BootStrap 不支持模态框重叠，不要在一个模态框上重叠另一个模态框。要想同时支持多个模态框，需要自己写额外的代码来实现。

在使用模态窗口时，可以通过 date 属性，或者通过 JavaScript 代码来实现。

1. 通过 data 属性

不需写 JavaScript 代码也可激活模态框。通过在一个起控制器作用的页面元素(例如，按钮)上设置 data - toggle="modal"，并使用 data - target="#foo"或 href="#foo"指向特定的模态框即可。

2. 通过 JavaScript 调用

只需一行 JavaScript 代码，即可通过 id myModal 调用模态框：

$('#myModal'). modal(options)

实现图 8 - 32 的模态窗口的代码片段如下。

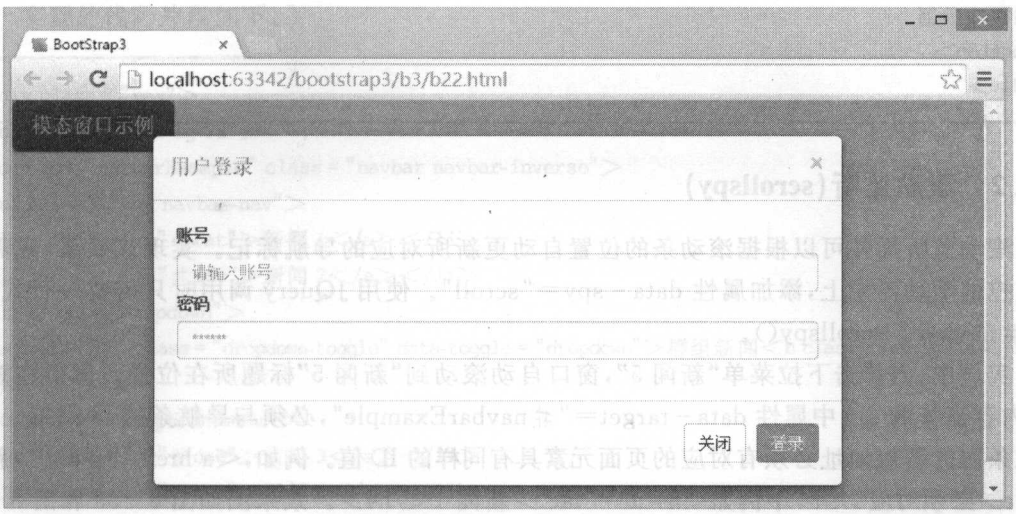


图 8-32　模态窗口效果

```
<div class = "container">
<div class = "modal fade" id = "myModal1"  role = "dialog"
aria-labelledby = "myModalLabel" aria-hidden = "true">
<div class = "modal-dialog">
<div class = "modal-content">
<div class = "modal-header">
<button type = "button" class = "close" data-dismiss = "modal" aria-hidden = "true">&times;
</button>
<h4 class = "modal-title">用户登录</h4>
</div>
<div class = "modal-body">
<form action = "#">
<label>账号</label>
<input class = "form-control" placeholder = "请输入账号"/>
<label>密码</label>
<input class = "form-control" type = "password" placeholder = "******"/>
</form>
</div>
<div class = "modal-footer">
<button type = "button" class = "btn btn-default" data-dismiss = "modal">关闭</button>
<button type = "button" class = "btn btn-primary">登录</button>
</div>
</div><!-- /.modal-content -->
</div><!-- /.modal-dialog -->
</div><!-- /.modal -->

</div>
<div class = "bs-example" style = "padding-bottom: 24px;">
<button class = "btn btn-primary btn-lg" data-toggle = "modal" data-target = "#myModal1"
        data-backdrop = "true" >
```

```
模态窗口示例
</button>
</div>
```

8.6.2 滚动监听(scrollspy)

滚动监听插件可以根据滚动条的位置自动更新所对应的导航标记。实现该效果,只需要在监控的滚动要素上,添加属性 data-spy="scroll"。使用 JQuery 调用时只需要一行代码:$('#navbar').scrollspy()。

实例中,当单击下拉菜单"新闻 5",窗口自动滚动到"新闻 5"标题所在位置。值得注意的是,内容窗体的 div 中属性 data-target="#navbarExample",必须与导航条的 id 保持一致。导航条内的链接地址必须有对应的页面元素具有同样的 ID 值。例如,新闻 1必须对应 DOM 中例如<h4 id="fat">新闻 1</h4>。效果图如图 8-33 和图 8-34 所示。

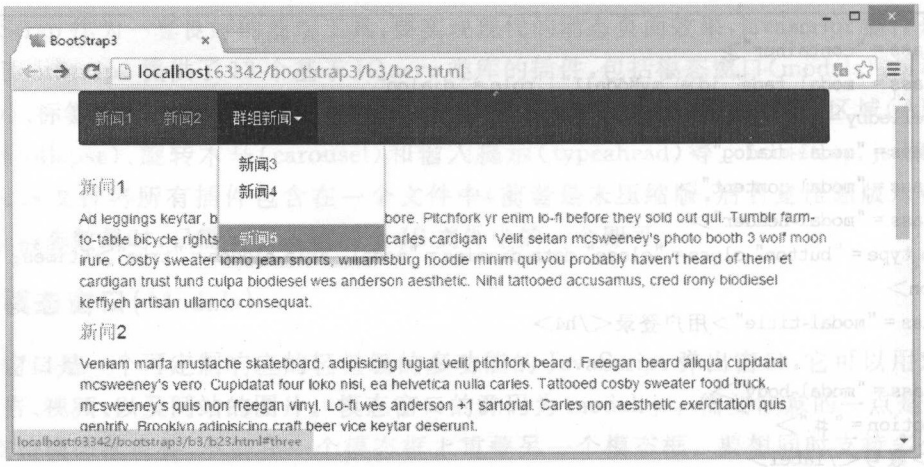


图 8-33 滚动监听效果图 1-单击前

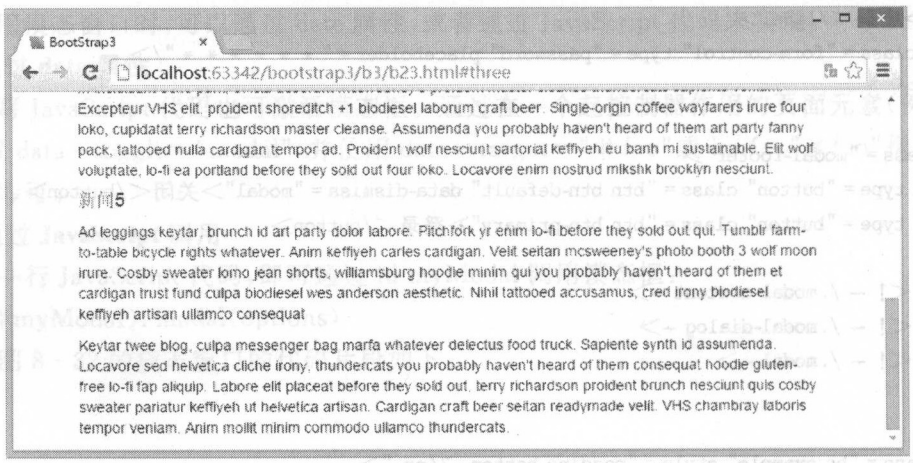


图 8-34 滚动监听效果图 2-单击后

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-12">
<div id = "navbarExample" class = "navbar navbar-inverse">
<ul class = "nav navbar-nav">
<li><a href = "#fat">新闻 1</a></li>
<li><a href = "#mdo">新闻 2</a></li>
<li class = "dropdown">
<a href = "#" class = "dropdown-toggle" data-toggle = "dropdown">群组新闻<b class = "caret"></b>
</a>
<ul class = "dropdown-menu">
<li><a href = "#one">新闻 3</a></li>
<li><a href = "#two">新闻 4</a></li>
<li class = "divider"></li>
<li><a href = "#three">新闻 5</a></li>
</ul>
</li>
</ul>

</div>
</div>
</div>
<div class = "row" style = "align-content: center">
<div class = "col-lg-1"></div>
<div class = "col-lg-10" data-spy = "scroll" data-target = "#navbarExample" style = "align-content:
            center">
<h4 id = "fat">新闻 1</h4>
<p>Ad leggings keytar, brunch id art party dolor labore. Pitchfork yr enim lo-fi before they sold out
qui. Tumblr farm-to-table bicycle rights whatever. Anim keffiyeh carles
            cardigan. Velit seitan mcsweeneys photo booth 3 wolf moon irure. Cosby
            sweater lomo jean shorts, williamsburg hoodie minim qui you probably haven't heard of
them et cardigan trust fund culpa biodiesel wes anderson aesthetic.
            Nihil tattooed accusamus, cred irony biodiesel keffiyeh artisan ullamco
            consequat.
</p>
<h4 id = "mdo">新闻 2</h4>
<p>Veniam marfa mustache skateboard, adipisicing fugiat velit pitchfork beard. Freegan beard aliqua
cupidatat mcsweeneys vero. Cupidatat four loko nisi, ea helvetica nulla
            carles. Tattooed cosby sweater food truck, mcsweeneys quis non freegan vinyl.
            Lo-fi wes anderson +1 sartorial. Carles non aesthetic exercitation quis gentrify.
            Brooklyn adipisicing craft beer vice keytar deserunt.
</p>
<h4 id = "one">新闻 3</h4>
```

```
<p>Occaecat commodo aliqua delectus. Fap craft beer deserunt skateboard ea. Lomo bicycle rights adipisicing banh mi, velit ea sunt next level locavore single-origin coffee
                in magna veniam. High life id vinyl, echo park consequat quis aliquip banh mi
                pitchfork. Vero VHS est adipisicing. Consectetur nisi DIY minim messenger bag.
                Cred ex in, sustainable delectus consectetur fanny pack iphone.
</p>
<h4 id = "two">新闻 4</h4>
<p>In incididunt echo park, officia deserunt mcsweeneys proident master cleanse thundercats sapiente veniam. Excepteur VHS elit, proident shoreditch + 1 biodiesel laborum
                craft beer. Single-origin coffee wayfarers irure four loko, cupidatat
                terry richardson master cleanse. Assumenda you probably haven't heard of them art
                party fanny pack, tattooed nulla cardigan tempor ad. Proident wolf nesciunt
                sartorial keffiyeh eu banh mi sustainable. Elit wolf voluptate, lo-fi ea portland
                before they sold out four loko. Locavore enim nostrud mlkshk brooklyn
                nesciunt.
</p>
<h4 id = "three">新闻 5</h4>
<p>Ad leggings keytar, brunch id art party dolor labore. Pitchfork yr enim lo-fi before they sold
                out qui. Tumblr farm-to-table bicycle rights whatever. Anim keffiyeh carles
                cardigan. Velit seitan mcsweeneys photo booth 3 wolf moon irure. Cosby
                sweater lomo jean shorts, williamsburg hoodie minim qui you probably haven't heard
                of them et cardigan trust fund culpa biodiesel wes anderson aesthetic.
                Nihil tattooed accusamus, cred irony biodiesel keffiyeh artisan ullamco
                consequat.
</p>
<p>Keytar twee blog, culpa messenger bag marfa whatever delectus food truck. Sapiente synth id
                assumenda. Locavore sed
                helvetica cliche irony, thundercats you probably haven't heard of them consequat hoodie gluten-free lo-fi fap aliquip. Labore elit
                placeat before they sold out, terry richardson proident brunch nesciunt quis
                cosby sweater pariatur keffiyeh ut helvetica artisan. Cardigan craft beer seitan readymade velit. VHS chambray laboris tempor veniam. Anim mollit minim
                commodo ullamco thundercats.
</p>

</div>
<div class = "col-lg-1"></div>
</div>
```

8.6.3 标签效果(tabs)

通过标签效果,可以实现单击标签,窗口内容跟随变化的动态效果,是当前网页中非常常见的一种内容变换方式。其实现方法与滚动监听类似。只需为页面元素简单的添加 data - toggle="tab"或 data - toggle="pill"就可以,无需写任何 JavaScript 代码也能激活标签页或

胶囊式导航。为 ul 添加. nav 和. nav - tabs classe 即可为其赋予 Bootstrap 标签页样式；而添加 nav 和 nav - pills class 可以为其赋予胶囊标签页样式。图 8 - 35 和图 8 - 36 为实现的标签效果图。

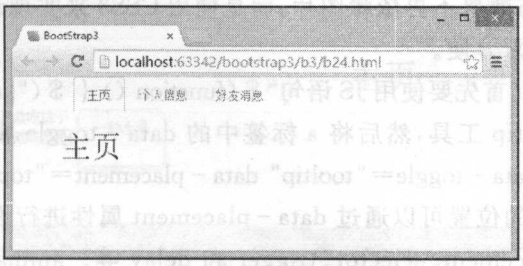


图 8 - 35　动态标签效果图 1 -单击标签前

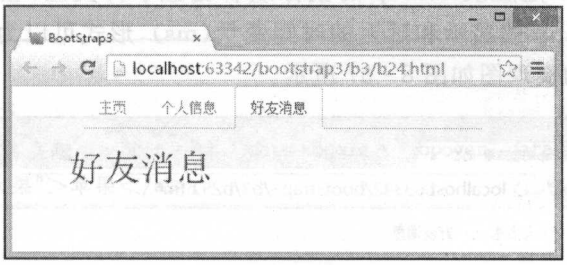


图 8 - 36　动态标签效果图 2 -单击标签后

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-12">
<ul class = "nav nav-tabs">
<li class = "active" ><a href = "#home" data-toggle = "tab">主页</a></li>
<li ><a href = "#info" data-toggle = "tab">个人信息</a></li>
<li ><a href = "#message" data-toggle = "tab">好友消息</a></li>
</ul>
</div>
<div class = "tab-content"  id = "mytab">
<div class = "tab-pane fade in active" id = "home">
<h1>主页</h1>
</div>
<div class = "tab-pane fade" id = "info">
<h1>个人信息</h1>
</div>
<div class = "tab-pane fade" id = "message"><h1>好友消息</h1></div>
</div>
</div>
</div>
```

8.6.4 提示效果(tooltip)

当用户鼠标划过或者放在某些链接上,会浮现的内容提示或者补充提示,这就是提示效果。BootStrap3 中的动态效果不再依赖图片,而是使用 CSS3 来实现动画效果,并使用 data 属性存储标题,使用起来非常方便。

如果想使用提示效果,首先要使用 JS 语句"$(function(){ $("[data-toggle='tooltip']").tooltip(); });"启动 tooltip 工具,然后将 a 标签中的 data-toggle 属性值设置为"tooltip"即可。如"橘子"。提示框出现的位置可以通过 data-placement 属性进行设置。提示框的其余属性选项包括:animation、placement、selector、trigger 和 delay 等。animation 是用来实现提示的淡出 css 效果;selector 选项就是提供给用户,用来控制 tooltip 出现在页面具体的某个目标上,默认是 false;trigger 是触发 tooltip 的鼠标或者键盘事件类型,包括 hover、focus、maual 等;delay 则是控制 tooltip 的显示和延迟的时间变量(ms),形式可以为:delay:{ show: 200, hide: 50}。提示效果的实现图如图 8-37 所示。

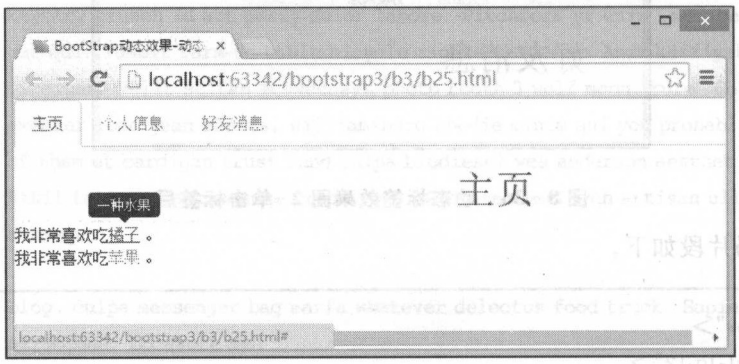


图 8-37 提示效果示例图

实现的代码片段如下。

```
<div class="row">
<div class="col-lg-6" >
<div id="ex">我非常喜欢吃<a href="#" data-toggle="tooltip" data-placement="top" title="
        一种水果">橘子</a>。
</div>
</div>
</div>
```

8.6.5 "泡芙"效果(popovers)

在 BootStrap 中,有一种与提示框类似的工具,叫做弹出框 popover,如果将上面的提示框工具代码中的 tooltip 改为 popover,则可以看到弹出框的效果,同样的,使用弹出框也需要 JS 语句"$(function(){ $("[data-toggle='popover']").popover();});"启动 popover 工具。

示例效果如图 8－38 所示。

图 8－38　弹出框效果示例图

实现的代码片段如下。

```
<div class = "row">
<div class = "col-lg-6" >
<div id = "ex2">我非常喜欢吃<a href = " # " data-toggle = "popover" data-placement = "right" title
        = "一种水果">苹果</a>。
</div>
</div>
</div>
```

8.6.6　折叠效果(collapse)

BootStrap3 的折叠插件可以很容易地让页面区域折叠起来。通过 data 属性:向元素添加 data-toggl 属性和 href 属性,自动分配可折叠元素的控制。为了向可折叠控件添加类似折叠面板的分组管理,需要添加 data 属性如 data-parent＝" # accordion "。实现的折叠效果图如图 8－39 和图 8－40 所示。

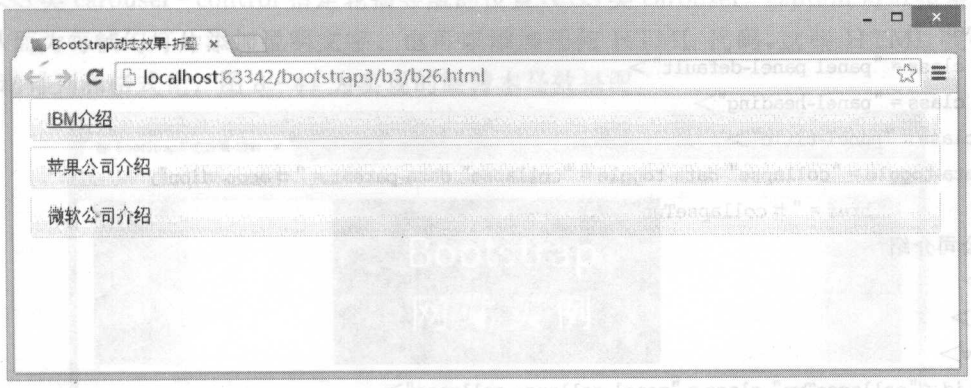


图 8－39　折叠效果图 1－单击标签前

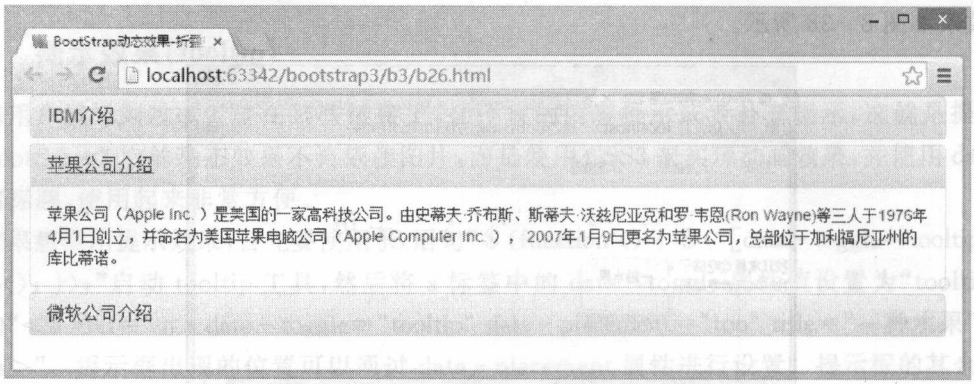


图 8-40　折叠效果图 2-单击标签后

实现的代码片段如下。

```
<div class = "col-lg-8">
<div class = "panel-group" id = "accordion">
<div class = "panel panel-default">
<div class = "panel-heading">
<h4 class = "panel-title">
<a data-toggle = "collapse" data-parent = "#accordion" href = "#collapseOne">
                    IBM 介绍
</a>
</h4>
</div>
<div id = "collapseOne" class = "panel-collapse collapse in">
<div class = "panel-body">
IBM,即国际商业机器公司,1911 年创立于美国,是全球性的信息技术和业务解决方案公司,业务遍及 170 多
        个国家和地区.2014 年全球营业收入达 928 亿美元,在美国注册了 7,534 项专利.在中国,IBM
        公司提供领先的云计算、大数据和分析、企业移动应用、社交商务、信息安全等解决方案.
</div>
</div>
</div>
<div class = "panel panel-default">
<div class = "panel-heading">
<h4 class = "panel-title">
<a data-toggle = "collapse" data-toggle = "collapse" data-parent = "#accordion"
            href = "#collapseTwo">
苹果公司介绍
</a>
</h4>
</div>
<div id = "collapseTwo" class = "panel-collapse collapse">
<div class = "panel-body">
苹果公司(Apple Inc. )是美国的一家高科技公司。由史蒂夫·乔布斯、斯蒂夫·沃兹尼亚克和罗·韦恩
        (Ron Wayne)等三人于 1976 年 4 月 1 日创立,并命名为美国苹果电脑公司(Apple Computer
```

```
Inc. ),2007 年 1 月 9 日更名为苹果公司,总部位于加利福尼亚州的库比蒂诺。
</div>
</div>
</div>
<div class = "panel panel-default">
<div class = "panel-heading">
<h4 class = "panel-title">
<a data-toggle = "collapse" data-toggle = "collapse" data-parent = "#accordion"
        href = "#collapseThree">
微软公司介绍
</a>
</h4>
</div>
<div id = "collapseThree" class = "panel-collapse collapse">
<div class = "panel-body">
微软(Microsoft),是一家总部位于美国的跨国电脑科技公司,是世界 PC(Personal Computer,个人计算机)机
        软件开发的先导,由比尔·盖茨与保罗·艾伦创办于 1975 年,公司总部设立在华盛顿州的雷德
        蒙德市(Redmond,邻近西雅图)。以研发、制造、授权和提供广泛的电脑软件服务业务为主。
</div>
</div>
</div>
</div>
</div>
```

8.6.7　旋转木马(carousel)

在网页中,常常看到一张张图片轮流播放的页面效果,俗称“旋转木马”,BootStrap 中的 carousel 插件能轻松帮实现这一效果。在页面中使用 CSS 类 carousel slide,即可使用旋转木马效果。CSS 类 carousel - indicators 指定轮播的指标,CSS 类 carousel - inner 指定轮播的项目,CSS 类 carousel - control 指定轮播导航的位置,CSS 类 carousel - caption 可以指定轮播标题,从而为每帧幻灯片添加说明文字。也可以添加任何 HTML 代码,这些 HTML 代码将会被自动排列和格式化。图 8 - 41 为实现的旋转木马效果图。

图 8 - 41　旋转木马效果图

实现的代码片段如下。

```
<div class = "col-lg-4">
<div id = "myCarousel" class = "carousel slide">
<! -- 轮播(Carousel)指标 -->
<ol class = "carousel-indicators">
<li data-target = "#myCarousel" data-slide-to = "0" class = "active"></li>
<li data-target = "#myCarousel" data-slide-to = "1"></li>
<li data-target = "#myCarousel" data-slide-to = "2"></li>
</ol>
<! -- 轮播(Carousel)项目 -->
<div class = "carousel-inner">
<div class = "item active">
<div align = "center">
<img  src = "img/expo.png">
<div class = "carousel-caption">BootStrap 网站案例</div>
</div>
</div>
<div class = "item">
<div align = "center">
<img src = "img/expo2.png" >
<div class = "carousel-caption">BootStrap 编码规范</div>
</div>
</div>
<div class = "item">
<div align = "center">
<img src = "img/expo3.png" >
<div class = "carousel-caption">BootStrap 招聘信息</div>
</div>
</div>
</div>
<! -- 轮播(Carousel)导航 -->
<a class = "carousel-control left" href = "#myCarousel"
        data-slide = "prev">&lsaquo;</a>
<a class = "carousel-control right" href = "#myCarousel"
        data-slide = "next">&rsaquo;</a>
</div>
</div>
```

8.6.8 附加导航(Affix)

附加导航插件允许某个 固定在页面的某个位置,最常见的就是 QQ 空间的左边导航部分。附加导航会锁定在某个位置,不会随着页面其他部分一起滚动。附加导航的实现,需要使用属性 data-spy="scroll"来定义附加导航,保持 data-target="#myScrollspy"与附加导航 id 一致,然后在导航上添加 data-spy="affix"属性即可。图 8-42 和图 8-43 所示为附

加导航效果图。


Bootstrap 附加导航（Affix）插件
第一部分
第二部分
第三部分
第四部分
第五部分
第一部分
这是第一部分，这是第一部分，这是第一部分，

```
</div>
<div class = "row">
<div class = "col-xs-3" id = "myScrollspy">
<ul class = "nav nav-pills nav-stacked" data-spy = "affix">
<li class = "active"><a href = "#section-1">第一部分</a></li>
<li><a href = "#section-2">第二部分</a></li>
<li><a href = "#section-3">第三部分</a></li>
<li><a href = "#section-4">第四部分</a></li>
<li><a href = "#section-5">第五部分</a></li>
</ul>
</div>
<div class = "col-xs-9">
<h2 id = "section-1">第一部分</h2>
<p>
这是第一部分,这是第一部分,这是第一部分,这是第一部分,这是第一部分这是第一部分,
        这是第一部分
</p>
<hr>
<h2 id = "section-2">第二部分</h2>
<p>这是第二部分,这是第二部分,这是第二部分,这是第二部分,这是第二部分,这是第二部分,
        这是第二部分,这是第二部分,这是第二部分,这是第二部分,这是第二部分,这是
        第二部分,这是第二部分,这是第二部分,这是第二部分,这
        是第二部分</p>
<hr>
<h2 id = "section-3">第三部分</h2>
<p>这是第三部分,这是第三部分,这是第三部分,这是第三部分,这是
        第三部分,这是第三部分,这是第三部分,这是第三部分,这是第三部分,这是
        第三部分,这是第三部分,这是第三部分,这是第三部分,这是第
        三部分,这是第三部分,这
        是第三部分,这是第三部分</p>
<hr>
<h2 id = "section-4">第四部分</h2>
<p>这是第四部分,这是第四部分,这是第四部分,这是第四部分,这是
        第四部分,这是第四部分,这是第四部分,这是第四部分,这
        是第四部分,这是第四部分,这是第四部分,</p>
<hr>
<h2 id = "section-5">第五部分</h2>
<p>这是第五部分,这是第五部分,这是第五部分,这是第五部分,这是
        第五部分,这是第五部分,这是第五部分,这是第五部分,这是第五部分,这
        是第五部分,这是第五部分,这是第五部分,这是第五部分,这是第
        五部分</p>
</div>
</div>
</div>
</body>
```

第9章

综合实训——博客信息系统

9.1 系统需求分析

博客系统是指使用计算机语言编写，方便用户使用的一个信息交流分享平台。本章节研究的就是在互联网上建立个人博客的一整套系统。

9.1.1 用例图

用例图主要用于描述用例、参与者以及它们之间的关系。用例代表一个完整的功能，UML 中的用例是动作步骤的集合。用例与参与者之间是关联关系，这种关联表明哪种参与者或角色能与该用例通信，此关系是双向的一对一关系。

博客管理涉及管理员、博主、访问者三种成员，因此本系统主要有 3 种角色：管理员、普通用户、匿名用户。这三种角色分别由不同的权限和功能，如用例图 9-1 所示。

系统管理员的权限：博客账户的添加、删除及修改；统计博客及评论和浏览次数；博客及评论的删除、查看；修改自己和普通用户密码；分类博客。

普通用户的功能：修改自己密码；博客的添加、查看、修改及删除；博客的搜索；评论的查看及回复管理。

匿名用户的功能：查看博客信息。

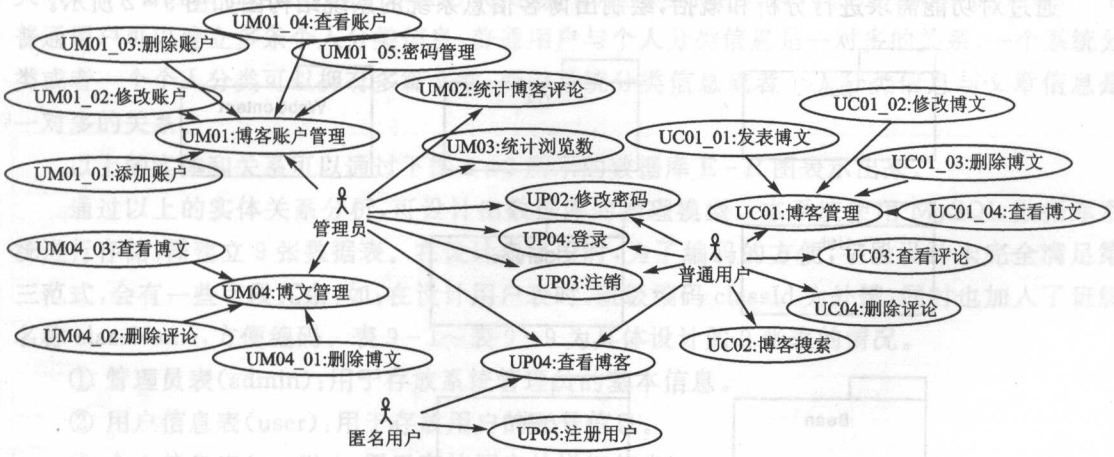


图 9-1 系统用例图

9.1.2 功能分析

根据以上分析，本系统应具备以下功能模块：

1. 系统设置模块

在处理业务或系统运行之前一般都需要设置一些基础数据，本系统主要是管理员信息。该信息由系统管理员进行设置，具体管理操作主要包括增、删、改、查、统计、分类等。

2. 删除评论

管理员及普通用户可以对博客的评论进行删除。

3. 删除博客

管理员及普通用户可以对博客进行删除。

4. 查看博客

管理员，普通用户及匿名用户可以对博客的内容、评论、浏览次数及发布日期进行查看。

5. 修改密码

系统管理员可以重置所有用户的密码，普通用户能修改自己的密码。

6. 博客信息管理

普通用户可以增加、修改博客及增加、回复评论。

9.2 系统架构

本系统的业务逻辑比较简单，容易理解，因此只需采用 JSP、Servlet、JavaBean 的 MVC 模式构建即可。本系统的表示层主要完成数据的显示、接收用户的输入等功能，使用 JSP 实现；HTML 用于数据的显示样式，JavaScript 负责用户客户端的数据验证；JavaBean 负责执行业务逻辑，封装对数据库表的操作，完成业务处理逻辑；Servlet 则作为逻辑控制器，接收 JSP 传递的用户请求，根据情况将信息转发给 JavaBean 处理，JavaBean 将处理结果返回给 Servlet，最后返回结果给客户端。

通过对功能需求进行分析和概括，绘制出博客信息系统的系统结构图如图 9-2 所示。

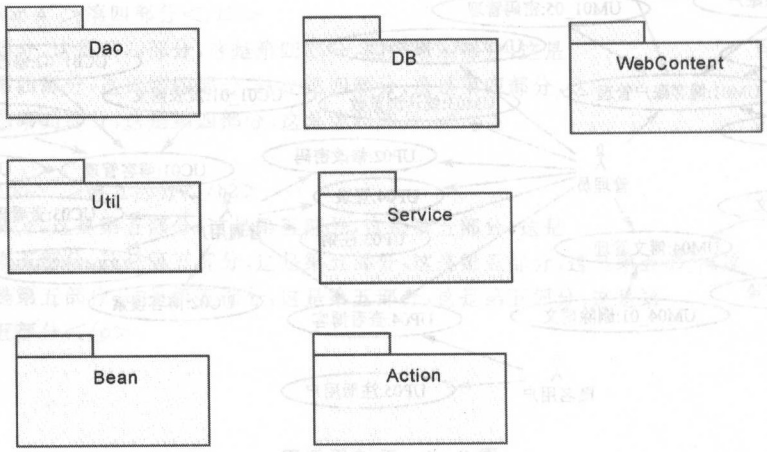


图 9-2 系统结构图

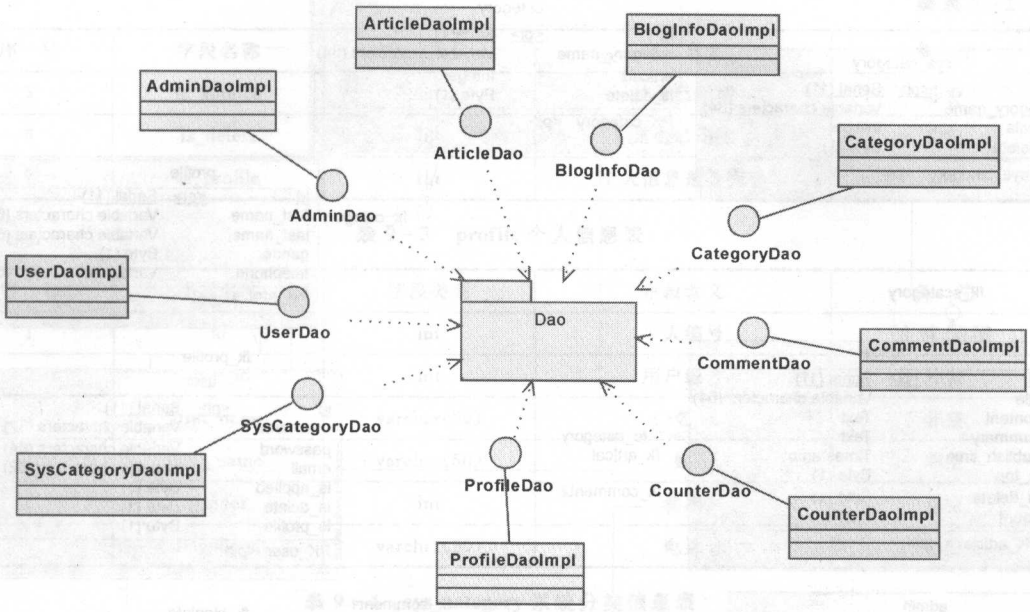


图 9-2　系统结构图(续)

9.3　数据库设计

根据功能描述,本系统有文章信息、评论信息、管理员、普通用户、匿名用户、系统分类信息、用户信息,个人分类信息等实体。一个管理员可以管理多个普通用户,管理员和用户之间是一对多的关系;一名普通用户可以发表多篇文章,普通用户与日志的关系是一对多;一篇文章可以有多个评论,因此文章与评论是一对多的关系;一个普通用户可以发表多个评论,普通用户与评论之间是一对多的关系;一个用户有一条个人信息,用户信息与个人信息是一对一的关系;一个管理员可以建立多个系统分类信息,管理员与系统分类信息是一对多的关系;一个普通用户可以建立多条个人分类信息,普通用户与个人分类信息是一对多的关系;一个系统分类或者一个个人分类可以拥有多篇文章,所以系统分类信息或者个人分类信息与文章信息是一对多的关系。

以上的实体和关系可以通过下图 9-3 所示的数据库 E-R 图表示出来。

通过以上的实体关系分析,可设计出数据库的物理模型。本系统使用 MySQL 数据库系统进行存储,需建立 9 张数据表。在设计表字段时,为了编码的方便,字段设计未完全满足第三范式,会有一些字段冗余,如:在设计用户表时,班级编码 classId 为外键,同时也加入了班级名称 className,方便编码。表 9-1～表 9-9 为具体设计的 9 张表的情况。

① 管理员表(admin):用于存放系统管理员的基本信息。

② 用户信息表(user):用于存放用户的账号信息。

③ 个人信息表(profile):用于存放用户的详细信息。

④ 个人博客信息表(blog_info):用于存放用户博客的详细信息。

⑤ 系统分类信息表(sys_category):用于存放系统文章分类信息。

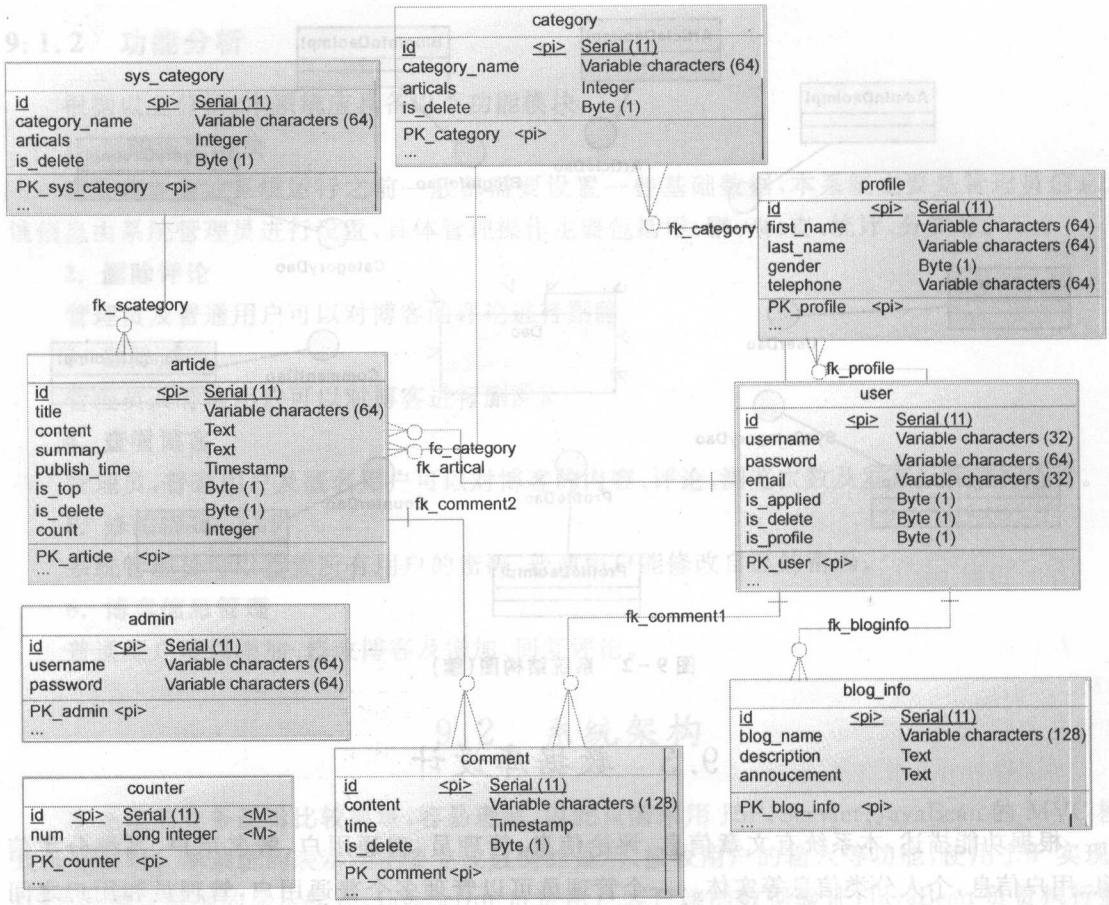


图 9-3　数据库 E-R 图

⑥ 个人分类信息表(category):用于存放用户文章分类信息。

⑦ 博客信息表(article):用于存放博客文章信息。

⑧ 统计信息表(counter):用于统计浏览网站的数量。

⑨ 评论信息表(comment):用于存放文章的评论信息。

表 9-1　admin 管理员表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	管理员编号	主键、自增
2	username	varchar(10)	管理员昵称	非空
3	password	varchar(128)	管理员密码	非空

表 9-2　user 用户信息表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	用户编号	主键、自增
2	username	varchar(50)	用户名	非空
3	password	varchar(16)	用户密码	非空
4	email	varchar(50)	邮箱	非空

续表 9－2

序　号	字段名称	字段类型	字段含义	备　注
5	is_applied	int	是否已激活	非空
6	is_delete	int	是否已删除	非空
7	is_profile	int	个人信息是否完善	非空

表 9－3　profile 个人信息表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	个人编号	主键、自增
2	user_id	int	用户编号	外键
3	first_name	varchar(50)	名	非空
4	last_name	varchar(50)	姓	非空
5	gender	int	性别	非空
6	telephone	varchar(50)	电话	非空

表 9－4　sys_category 系统分类信息表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	系统分类编号	主键、自增
2	category_name	varchar(50)	系统分类名	非空
3	articals	int	分类文章总数	非空
4	is_delete	int	是否删除	非空

表 9－5　category 个人分类信息表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	个人分类编号	主键、自增
2	user_id	int	用户编号	外键
3	category_name	varchar(50)	个人分类名	非空
4	articals	int	个人文章数量	非空
5	is_delete	int	是否删除	非空

表 9－6　blog_info 博客信息表

序　号	字段名称	字段类型	字段含义	备　注
1	blog_id	int	博客编号	主键、自增
2	User_id	int	用户编号	外键
3	blogName	varchar(100)	博客名	非空
4	description	varchar(1000)	描述	
5	propagate	vachar(500)	宣传	

表 9-7　article 文章信息表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	文章编号	主键,自增
2	user_id	int	用户编号	外键
3	sys_category_id	int	系统分类编号	外键
4	category_id	int	个人分类编号	外键
5	title	varchar(60)	标题	非空
6	content	mediumtext	内容	非空
7	summary	mediumtext	文章摘要	非空
8	publish_time	timestamp	发表时间	非空
9	is_top	int	是否置顶	非空
10	is_delete	int	是否删除	非空
11	count	int	文章点击数	非空

表 9-8　counter 统计信息表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	统计编号	主键
2	num	int	统计网站浏览数量	非空

表 9-9　comment 评论信息表

序　号	字段名称	字段类型	字段含义	备　注
1	id	int	评论编号	主键、自增
2	user_id	int	用户编号	外键
3	artical_id	int	文章编号	外键
4	content	varchar(1000)	评论内容	非空
5	time	date	评论时间	非空
6	is_delete	int	是否删除	非空

9.4　公共类设计

根据系统的总体设计和数据库设计,首先设计用于数据库连接和操作的公共类。本系统采用第 5 章介绍的连接池技术操作数据库。

① 创建 Dynamic Web Project 工程,工程名为:blog,创建好的工程结构图如图 9-4 所示。

② 将 MySQL 数据库的驱动程序复制到 WEB-INF/lib 目录,然后在 META-INF 目录下创建 context.xml,添加如下代码。

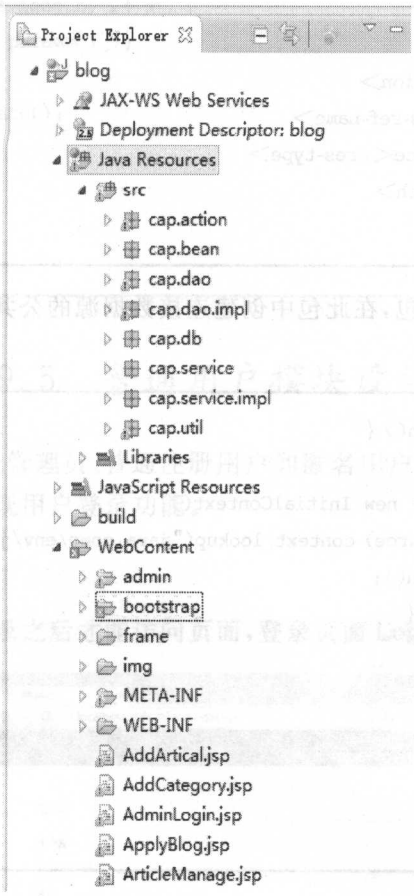


图 9-4　ScoreManage 工程结构图

```
<? xml version = "1.0" encoding = "UTF-8"? >
<context >
   <Resource  name = "jdbc/blog"
      auth = "Container"
      type = "javax.sql.DataSource"
      username = "root"
      password = "admin"
      maxActive = "100"
      maxIdle = "10"
      maxWait = "5000"
      URIEncoding = "UTF-8"
      driverClassName = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost:3306/blog? useUnicode = true&amp;characterEncoding = UTF-8"
   />
</context>
```

③ 在 web.xml 的<web-app></web-app>标签中添加下面的代码。

```
<resource-ref>
<description>blog</description>
<res-ref-name>jdbc/blog</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

④ 在 src 中创建 cap.db 包，在此包中创建连接数据源的公共类 DBPool.java，编辑如下代码，获取数据库连接。

```
public Connection getConnection() {
  try{
      InitialContext context = new InitialContext();
      DataSource ds = (DataSource) context.lookup("java:comp/env/jdbc/blog");
      conn  = ds.getConnection();
  } catch (NamingException e) {
      e.printStackTrace();
  } catch (SQLException e) {
      e.printStackTrace();
  }
  returnconn;
}
```

⑤ DBPool 类封装了查询功能的封装，doQueryRS()的输入参数为查询语句 sql 和参数对象列表 params，返回查询结果的对象列表，代码如下。

```
public ResultSet doQueryRS(String sql, Object[] params) throws SQLException {
  conn  = this.getConnection();
  PreparedStatement pstmt = conn.prepareStatement(sql);
  for (int i = 0; i < params.length; i++) {
      pstmt.setObject(i + 1, params[i]);
  }
  ResultSet rs = pstmt.executeQuery();
  return rs;
}
```

⑥ DBCon 类也封装了更新功能的封装，doUpdate()的输入参数为查询语句 sql 和参数对象列表 params，返回数据库更新的记录数量，代码如下。

```
public int doUpdate(String sql, Object[] params) throws SQLException {
  conn  = this.getConnection();
  PreparedStatement pstmt = conn.prepareStatement(sql);
```

```
    for (int i = 0; i < params.length; i++) {
        pstmt.setObject(i + 1, params[i]);
    }
    int res = pstmt.executeUpdate();
    if(conn! = null)
    conn.close();
    return res;
}
```

9.5　普通用户模块设计

本系统有 3 类用户：系统管理员、普通注册用户和匿名用户，前两者都需要登录进行相关功能的操作。接下来首先实现用户登录功能。

9.5.1　登录功能

本系统所有用户必须登录之后才能访问页面，登录页面 Login.jsp 的设计如图 9-5 所示。

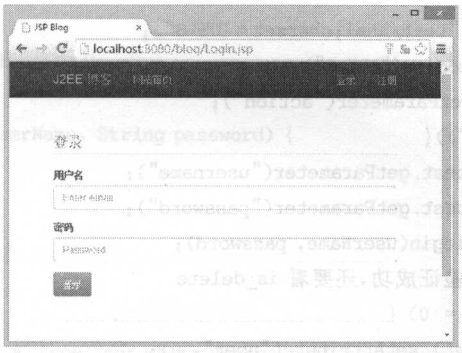


图 9-5　登录页面

用户名和密码的非空验证通过 javascript 进行客户端验证，实现的代码如下。

```
<script type = "text/javascript">
function isValidate(login_form) {
    var username = login_form.username.value;
    var password = login_form.password.value;
    if(username = = "" || password = = "") {
        alert("请填写用户名和密码!");
        returnfalse;
    }
    returntrue;
}
</script>
```

单击提交按钮，将用户名、密码、数据提交给 Servlet，用户登录的 Servlet 由 UserServlet.java 处理，利用 Servlet3.0 的注解@WebServlet 配置 url 为/login，则无需在 web.xml 中进行配置。核心代码如下。

```
package cap.action;
//省略导入包列表
@WebServlet("/user")
public class UserServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
  private UserService userService;
  public UserServlet() {
    userService = new UserServiceImpl();
  }
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    this.doPost(request, response);
  }
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    response.setCharacterEncoding("UTF-8");
    String action = request.getParameter("action");
    if(action.equals("login")){
      String userName = request.getParameter("username");
      String password = request.getParameter("password");
      User u = userService.login(userName, password);
      if (null != u) {  //验证成功，还要看 is_delete
      if (u.getIsDelete() == 0) {
          request.getSession().setAttribute("user", u);
          response.sendRedirect("user? action=index");
      } else {
          request.getSession().setAttribute("userIsDeleMsg", "该用户已被禁用，无法登录!");
          response.sendRedirect("Login1.jsp");
      }

    } else {
          request.getSession().setAttribute("msg", "验证失败，请重新输入用户名或密码!");
          response.sendRedirect("Login.jsp");
      }
}
```

User.java 类负责用户信息的存储，放在 cap.bean 包，代码如下。

```
public class User {
  privateintid;
```

```
  private String email;
  private String userName;
  privateintisApplied;
  privateintisDelete;
  privateintisProfile;
  //省略 getters 和 setters
}
```

对系统管理员信息的增、删、改、查操作是由 UserDAO 接口定义，其实现类是 AdminDaoImpl.java，放在 cap.dao 以及 cap.dao.impl 包，接口的定义如下。

```
public interface UserDao {
  publicabstract User login(String userName, String password);
}
```

AdminDaoImpl.java 的实现类如下。

```
package cap.dao.impl;
// 省略导入包列表
public class UserDaoImpl implements UserDao {
  @Override
  public User login(String userName, String password) {
    User u = null;
    ResultSet rs = null;
    DBPool dbc = new DBPool();
    try {
        u = new User();
        String sql = "select * from user where username = ? and password = ?";
        rs = dbc.doQueryRS(sql, new Object[]{userName,password});
        if (rs.next()) {
            u.setId(rs.getInt("id"));
            u.setEmail(rs.getString("email"));
            u.setUserName(rs.getString("username"));
            u.setIsApplied(rs.getInt("is_applied"));
            u.setIsDelete(rs.getInt("is_delete"));
            u.setIsProfile(rs.getInt("is_profile"));
        } else {
            u = null;
        }
        } catch (SQLException e) {
        e.printStackTrace();
        } finally {
            dbc.close();
        }
```

```
        return u;
    }
    }
```

9.5.2 文章查看及分页模块

当用户输入正确的用户名和密码之后，进入博客的首页，可以查看不同用户发布的博文，查看界面如图 9-6 所示。

图 9-6 博客查看页面

① 需要在 cap. dao 包中创建 ArticleDAO 接口，添加分页显示的方法，实现的关键代码如下。

```
public interface ArticleDao {
  public abstract List<Article> getArticleByPage(int curPage, int size);
}
```

② 编写 ArticleDAO 的实现类 ArticleDAOImpl. java，并添加如下实现代码。

```
@Override
public List<Article> getArticleByPage(int curPage, int size) {
  ResultSet rs = null;
  List<Article> artList = null;
  int start = (curPage - 1) * size;
```

```
  DBPool dbc = new DBPool();
    try {
    String sql = "select a. * ,u.username from article as a, user as u where u. id = a.user_id and a. is_
delete = 0 order by publish_time DESC limit ?,?";
          artList = new ArrayList<Article>();
    rs = dbc.doQueryRS(sql, new Object[]{start,size});
    while (rs.next()) {
      Article art = new Article();
      art.setId(rs.getInt("id"));
      art.setTitle(rs.getString("title"));
      art.setUserId(rs.getInt("user_id"));
      art.setSysCategoryId(rs.getInt("sys_category_id"));
      art.setCategoryId(rs.getInt("category_id"));
      art.setContent(rs.getString("content"));
      art.setSummary(rs.getString("summary"));
      art.setPublishTime(rs.getTimestamp("publish_time"));
      art.setIsTop(rs.getInt("is_top"));
      art.setIsDelete(rs.getInt("is_delete"));
      art.setCount(rs.getInt("count"));
      try {
        art.setUsername(rs.getString("username"));
      } catch (Exceptione) {
        art.setUsername("");
      }
      artList.add(art);
    }
    } catch (Exceptione) {
        e.printStackTrace();
    } finally {
        dbc.close();
    }
  return artList;
}
```

③ 由于要实现分页显示，就在 cap. util 中添加分页类 PageControl. java，实现的代码如下。

```
package cap.util;
import java.util.List;
public class PageControl {
  privateintcurPage  = 1;
  privateintpageSize;
  privateinttotalRows;
```

```
  privateinttotalPages;
  private List list;//用于存放分页数据
  public PageControl(String curPageStr, int totalRows) {
    if (null != curPageStr) {
        this.curPage = Integer.parseInt(curPageStr);  //初始化当前页数
  }
    this.totalRows = totalRows;//初始化总行数
    this.pageSize = 5;//设置每页显示的记录数
    //计算总页数
    this.totalPages = (this.totalRows / this.pageSize) + ((this.totalRows % this.pageSize) >
0 ? 1 : 0);
    }
  public List getList() {
    returnlist;
  }
  public int getCurPage() {
    returncurPage;
  }
  publicvoid setCurPage(int curPage) {
    this.curPage = curPage;
  }
  public int getPageSize() {
    returnpageSize;
  }
  publicvoid setPageSize(int pageSize) {
    this.pageSize = pageSize;
  }
  public int getTotalPages() {
    returntotalPages;
  }
  publicvoid setTotalPages(int totalPages) {
    this.totalPages = totalPages;
  }
  publicvoid setList(List list) {
    this.list = list;
  }
}
```

④ 创建 Article 的的服务接口 ArticleService，并添加如下的代码实现分页显示。

```
public interface ArticleService {
  public abstract PageControl getData(String curPageStr);
}
```

⑤ 在 ArticleServiceImpl.java 中实现 ArticleService 接口分页查询方法，实现代码如下。

```
package cap.service.impl;
public class ArticleServiceImpl implements ArticleService {
  private ArticleDao artDao;
  public ArticleServiceImpl() {
    artDao = new ArticleDaoImpl();
  }
  @Override
  public PageControl getData(String curPageStr){
    int count = artDao.getAllArtical().size();
    PageControl pc = new PageControl(curPageStr, count);
    List<Article> artList = artDao.getArticleByPage(pc.getCurPage(), pc.getPageSize());
    pc.setList(artList);
    return pc;
  }
}
```

最后在 UserServlet.java 中的 doPost 方法中添加下面的关键代码。

```
elseif(action.equals("index")){
            //获取系统分类列表
            List<SysCategory> scList = scService.getAllSysCategory();
            //获取 2 个活跃人数
            List<User> uList = artService.getActiveUser(2);
            //获取点击率前 10 的博文
            List<Article> tenList = artService.topTenArticle();
            //分页
            String curPageStr = request.getParameter("curPage");
            PageControl pc = artService.getData(curPageStr);
            request.setAttribute("curPage", pc.getCurPage());
            request.setAttribute("totalPages", pc.getPageSize());
            request.setAttribute("uList", uList);
            request.setAttribute("scList", scList);
            request.setAttribute("tenList", tenList);
            request.setAttribute("artList", pc.getList());
            request.getRequestDispatcher("/index.jsp").forward(request, response);
  }
```

9.5.3　文章管理模块

当用户单击博客管理后，会弹出下拉菜单，管理页面如图 9-7 所示。

在创建好的 ArticleServlet.java 中，在 doPost 方法中添加下面的关键代码。

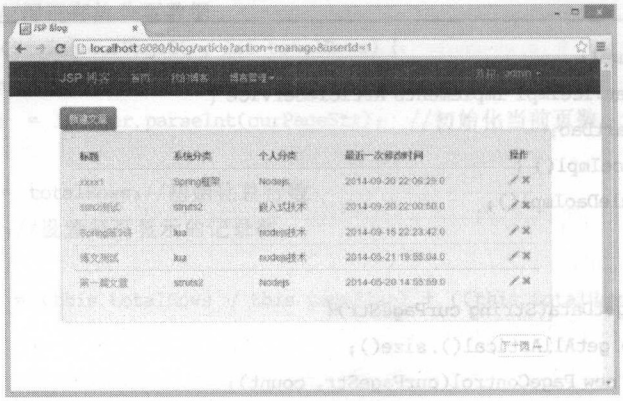


图 9-7　博客管理页面

```
elseif(action.equals("manage")){
   int userId = Integer.parseInt(request.getParameter("userId"));
   //添加用于分页
   String curPageStr = request.getParameter("curPage");
   PageControl pc = artService.getByPageUserId(curPageStr, userId);
   request.setAttribute("curPage", pc.getCurPage());
   request.setAttribute("totalPages", pc.getTotalPages());
   request.setAttribute("artList", pc.getList());
   request.getRequestDispatcher("ArticleManage.jsp").forward(request, response);
}
```

具体的实现代码和前一个模块实现原理相似，唯一的区别就是根据用户的主键查询出文章列表并分页显示。具体的实现请参考随书提供的源代码。

9.5.4　文章发布模块

单击图 9-7 中的新建文章，新建文章的页面设计如图 9-8 所示。

图 9-8　文章页面

① 在 ArticleServlet 中添加如下的代码。

```
elseif(action.equals("save")){
  int userId = Integer.parseInt(request.getParameter("userId"));
  String title = request.getParameter("title");
  int scgId = Integer.parseInt(request.getParameter("sys_category"));
  int cgId = Integer.parseInt(request.getParameter("category"));
  String summary = request.getParameter("summary");
  String content = request.getParameter("content");
  int res = artService.insertArtical(title, userId, scgId, cgId, content, summary);

  if (res > 0) {     //添加新文章成功
    request.getSession().setAttribute("succMsg", "发布文章成功!");
  } else {
    request.getSession().setAttribute("errorMsg", "发布文章失败");
  }
  response.sendRedirect("article? action=manage&userId=" + userId);
}
```

② 在 ArticleDAO 接口中添加如下的方法，具体的定义如下面的代码。

```
public interface ArticleDao {
  public abstract int insertArticle(String title, int userId, int scgId,
  int cgId, String content, String summary);
}
```

③ 接着在 ArticleDAO 的实现类 ArticleDAOImpl.java 实现，实现的关键代码如下。

```
@Override
public int insertArticle(String title, int userId, int scgId, int cgId, String content, String summary) {
  int res = -1;
  DBPool dbc = new DBPool();
  try {
    String sql = "insert into article values(NULL, ?, ?, ?, ?, ?, ?, NOW(), 0, 0)";
  res = dbc.doUpdate(sql, new Object[]{title,userId,scgId,cgId,content,summary});
  } catch (SQLException e) {
    e.printStackTrace();
  } finally {
    dbc.close();
  }
  return res;
}
```

本节中实现了博客系统的登录、博文的查看、博文的管理、博文的发布。由于篇幅的限制，具体的实现代码没有在此体现完全，只涉及了普通用户模块。此外还有匿名用户，管理用户模块没有讲解，请读者查阅随书的源代码，以便更进一步的学习。

附录 A

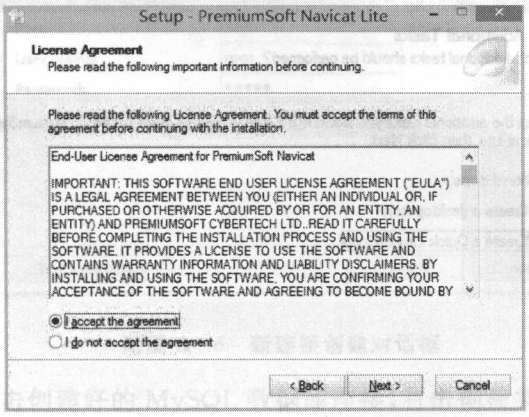
Navicat 的安装使用

Navicat for MySQL 是一套专为 MySQL 设计的高性能数据库管理及开发工具。它可以用于任何版本 3.21 或以上的 MySQL 数据库服务器，并支持大部份 MySQL 最新版本的功能，包括触发器、存储过程、函数、事件、视图、管理用户等。下面介绍一下 Navicat Lite for MySQL 的使用方法。

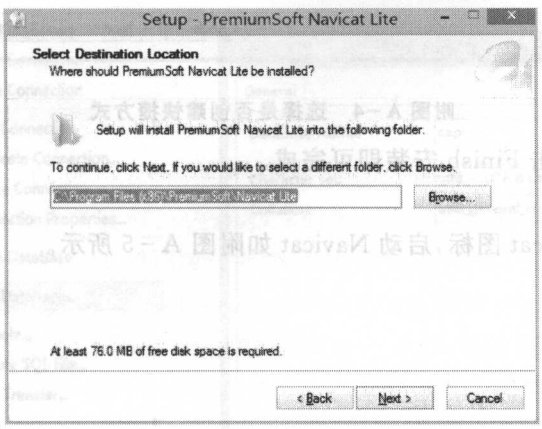
1. 软件的下载与安装

① 下载软件，软件下载地址：http://www.navicat.com.cn/download。

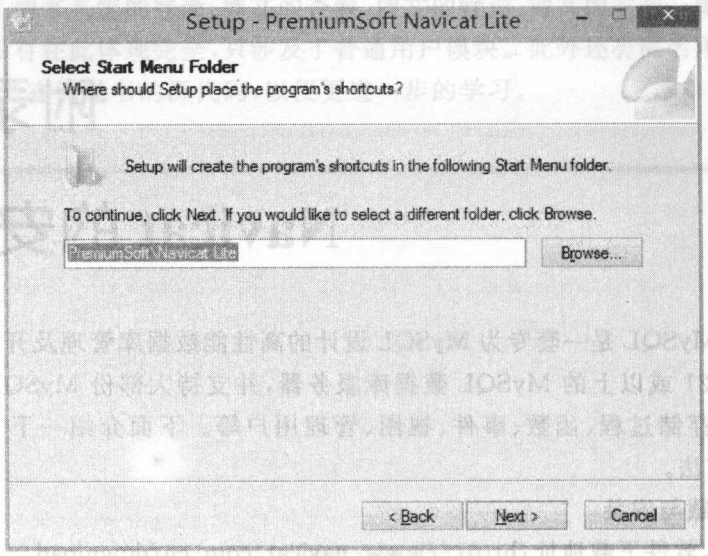
② 下载完成后，解压文件夹，单击解压后的文件开始进行如附图 A-1～附图 A-4 所示的安装步骤。



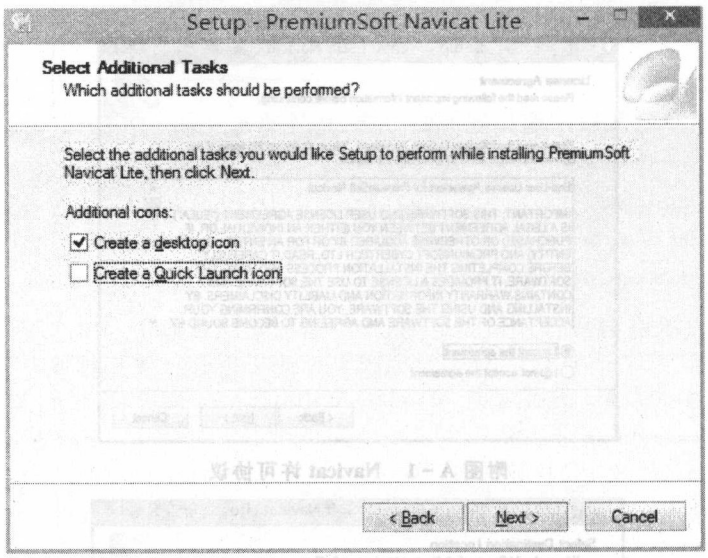
附图 A-1　Navicat 许可协议



附图 A-2　选择安装路径



附图 A-3　创建快捷方式的路径

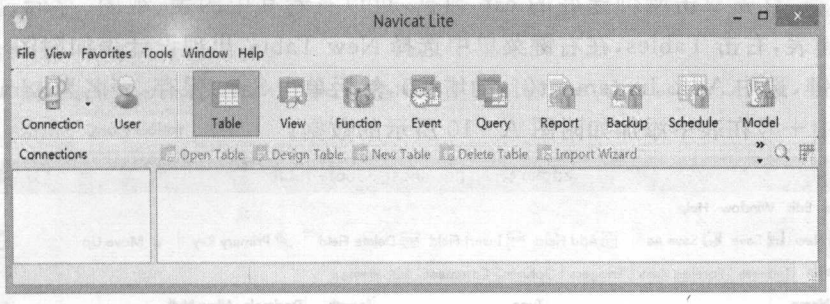


附图 A-4　选择是否创建快捷方式

③ 单击下一步，单击 Finish 安装即可完成。

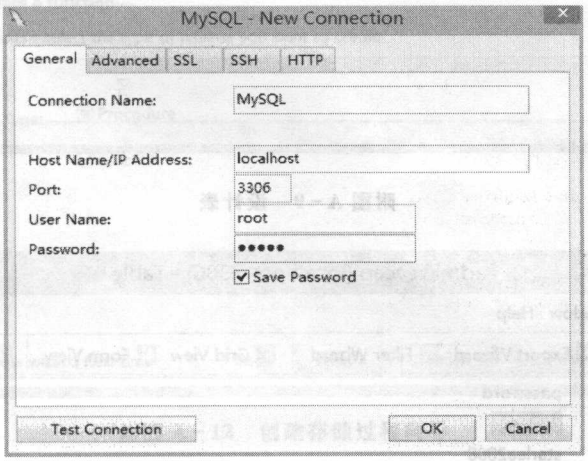
2. 软件的使用说明

双击桌面上的 Navicat 图标，启动 Navicat 如附图 A-5 所示。



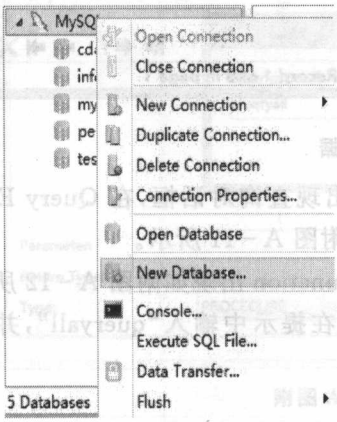
附图 A－5　Navicat 主界面图

① 创建连接：单击 File/New Connection/MySQL，可以进行创建 MySQL 连接，如附图 A－6 所示。输入连接名为 MySQL，连接密码，先单击左下角的 Test Connection，如果出现 Connection Successful 的提示框，说明 MySQL 数据库连接成功，然后单击 OK。

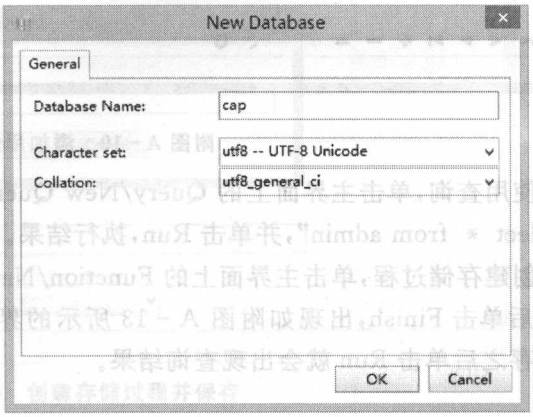


附图 A－6　新连接创建对话框

② 创建数据库：双击创建好的 MySQL 数据库连接，右击创建案例中所需的数据库，如附图 A－7 所示。接着在出现的创建数据库的对话框中进行输入，如附图 A－8 所示。

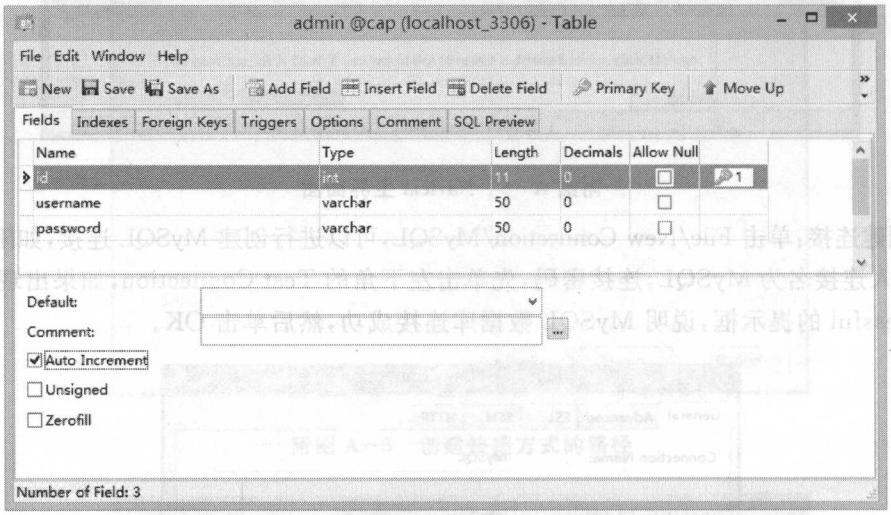


附图 A－7　创建新数据库菜单

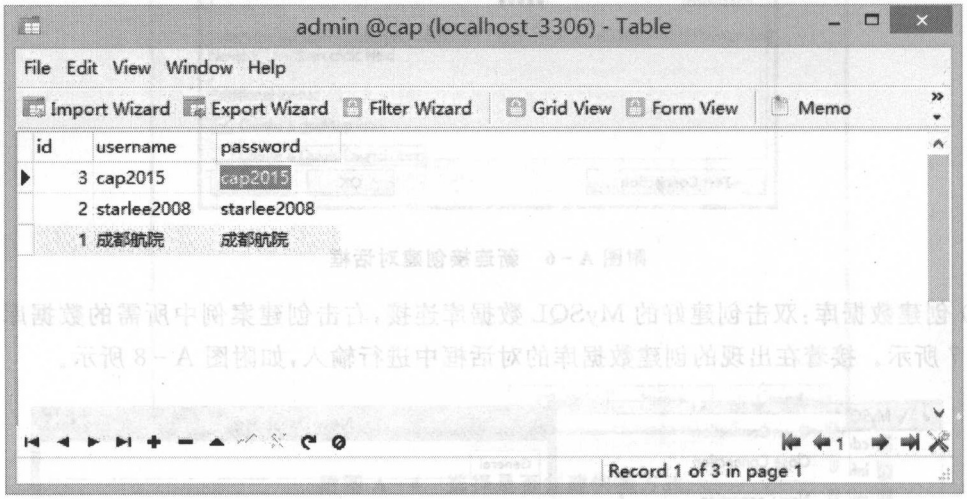


附图 A－8　创建数据库对话框

③ 创建表：首先双击刚创建好的 cap 数据，可以查看其中的表、视图、存储过程等信息。其次开始创建表，右击 Tables，在右键菜单中选择 New Table，出现设计表如附图 A－9 所示。其中 id 为主键，选中 Auto Increment(自动增长)，然后单击 Save 保存，表名为 admin；双击 admin，单击图中＋号在表中添加如附图 A－10 所示的数据。



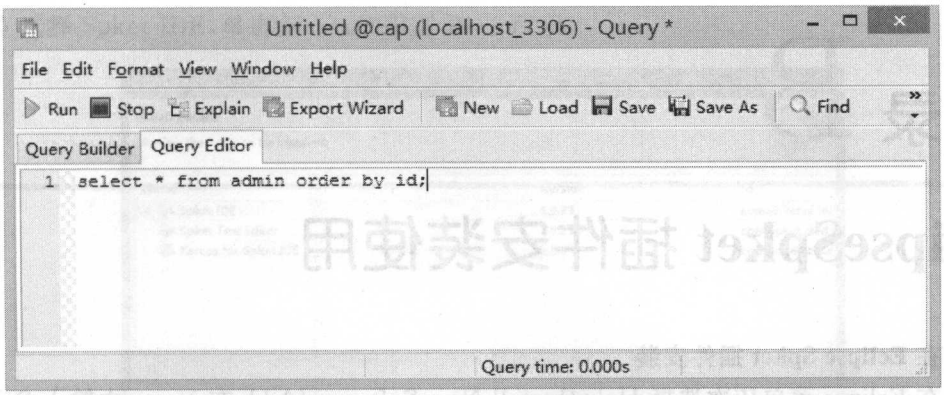
附图 A－9　设计表



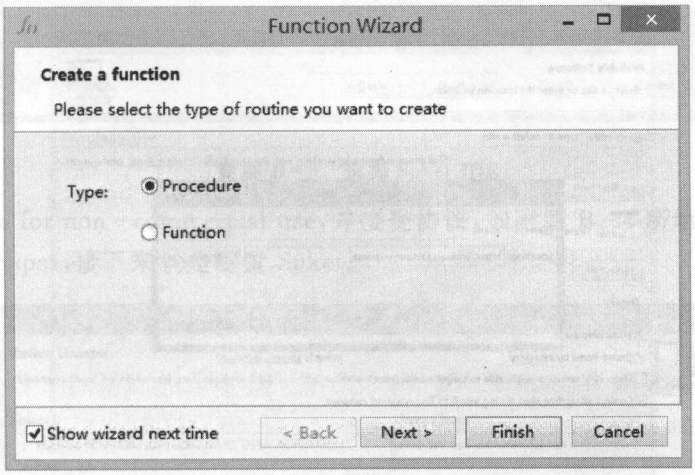
附图 A－10　添加用例数据

④ 使用查询，单击主界面上的 Query/New Query，出现查询对话框，在 Query Editor 中输入“select ＊ from admin”，并单击 Run，执行结果。如附图 A－11 所示。

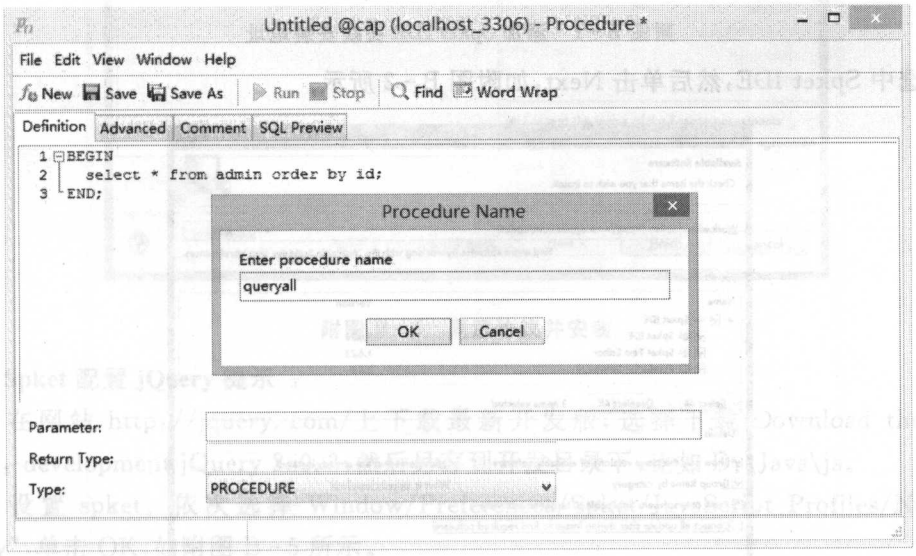
⑤ 创建存储过程，单击主界面上的 Function/New Function 出现如附图 A－12 所示的对话框，然后单击 Finish，出现如附图 A－13 所示的界面，在提示中输入“queryall”，并且单击 Save，保存之后单击 Run 就会出现查询结果。



附图 A－11　查询结果



附图 A－12　创建存储过程向导



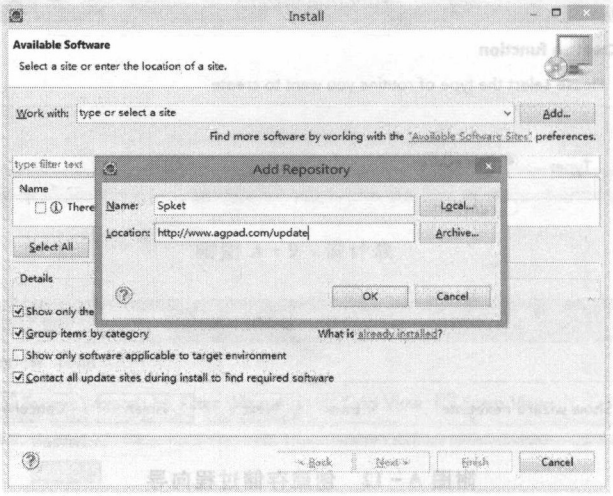
附图 A－13　创建存储过程并保存

附录 B

EclipseSpket 插件安装使用

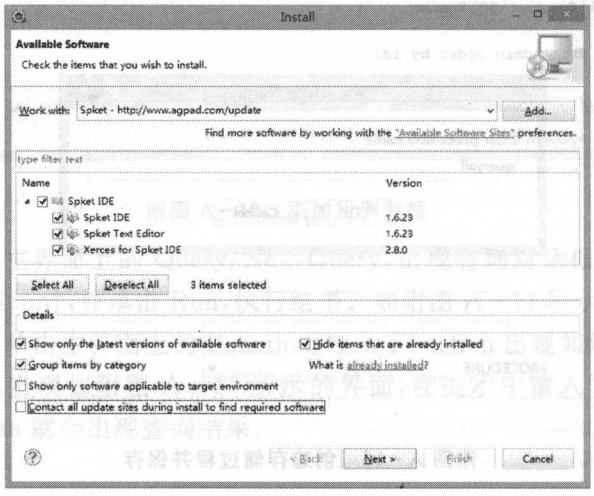
1. 在 Eclipse Spket 插件安装

① 在 Eclipse 菜单依次选择 Help/Install New Software/Add，在 Name 中输入 Spket，在 Location 中输入 http://www.agpad.com/update。如附图 B-1 所示。



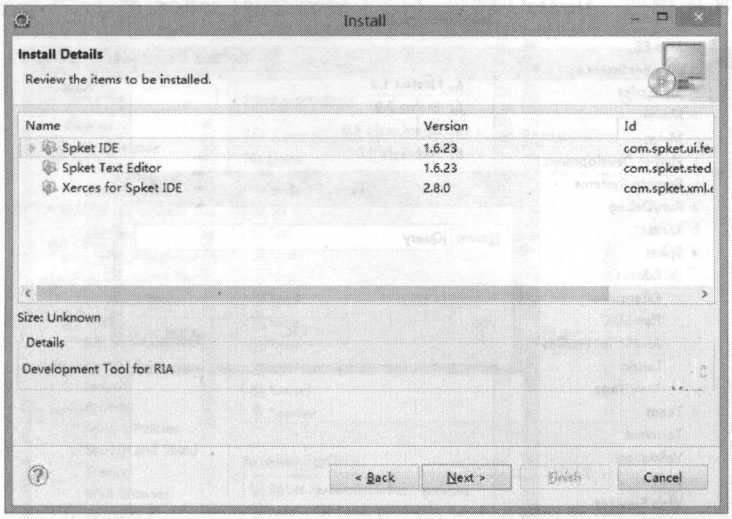
附图 B-1　添加 Spket IDE 在线安装地址

② 选中 Spket IDE，然后单击 Next，如附图 B-2 所示。



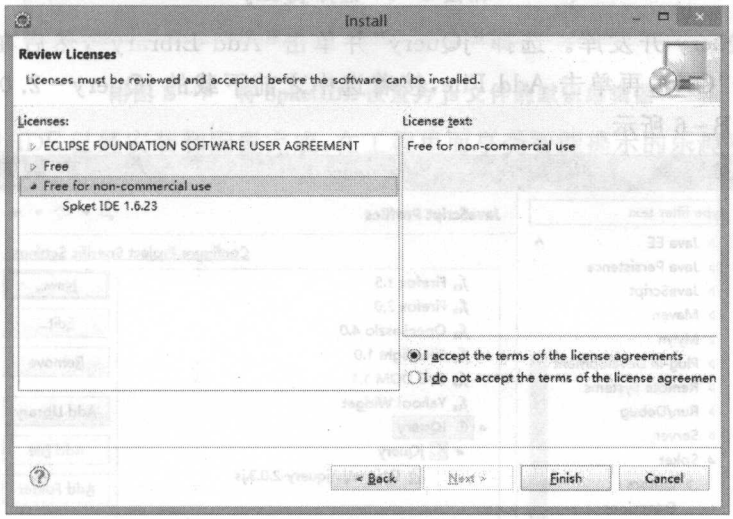
附图 B-2　选择需要安装的内容

③ 选择 Spket IDE，单击 Next，如附图 B－3 所示。



附图 B－3　选择 Spket IDE

④ 选择 Free for non－commercial use，并接受协议，如附图 B－4 所示。单击 Finish 后，安装完成重启 Eclipse，接下来就是配置 Spket。

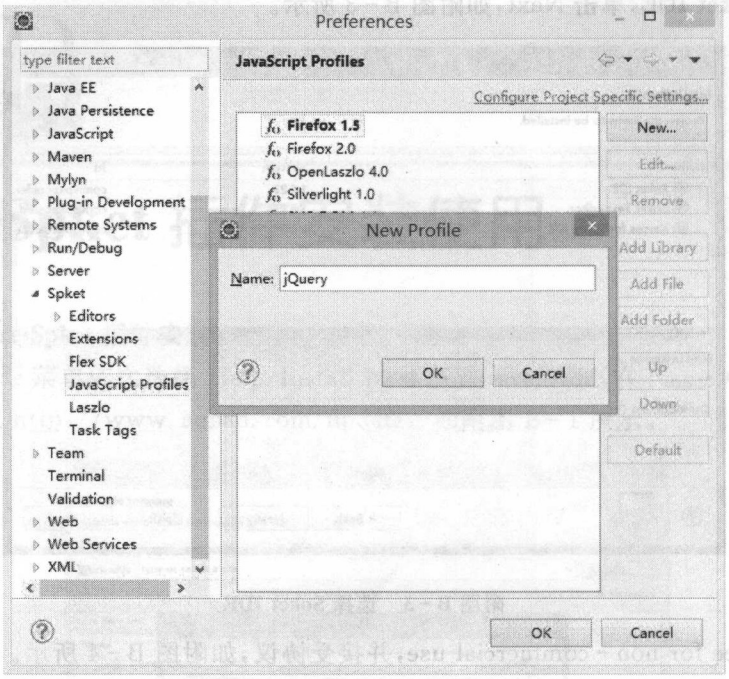


附图 B－4　同意协议并安装

2. Spket 配置 jQuery 提示

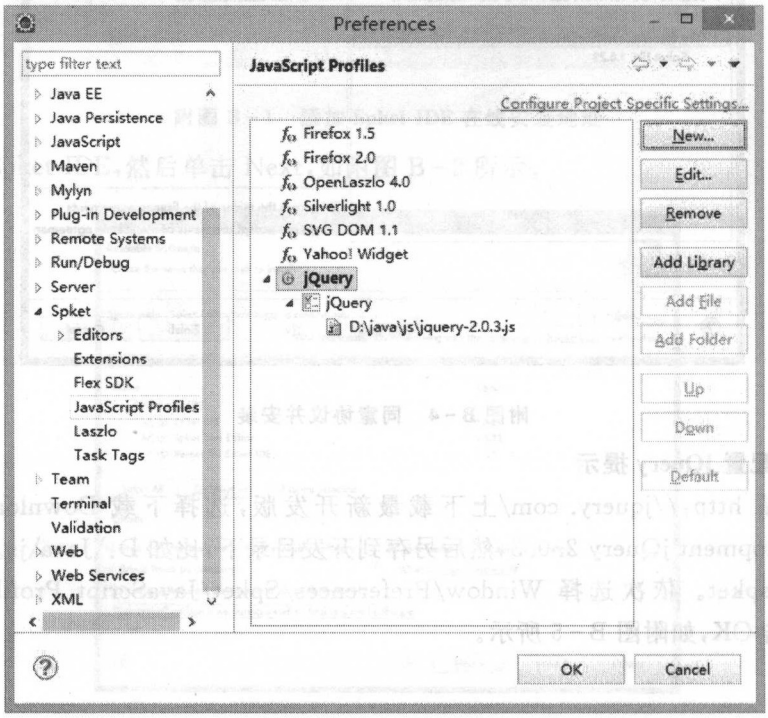
① 在网站 http://jquery.com/上下载最新开发版，选择下载 Download the uncompressed, development jQuery 2.0.3，然后另存到开发目录下，比如 D:\Java\js。

② 设置 spket。依次选择 Window/Preferences/Spket/JavaScript Profiles/New，输入“jQuery”，单击 OK，如附图 B－5 所示。



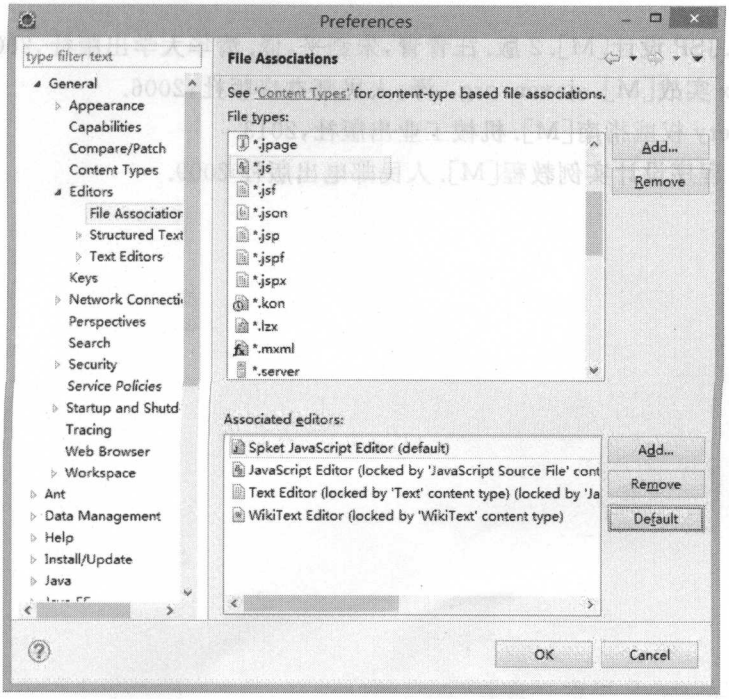
附图 B-5　选择 jQuery

③ 添加 jQuery 开发库。选择“jQuery”并单击“Add Library”，然后在下拉条中选取“jQuery”并单击“OK”，再单击 Add File，接着选中之前下载的 jQuery-2.0.3.js 文件设成 Default，如附图 B-6 所示。



附图 B-6　添加 jQuery 开发库

④ 设置 js 打开方式。在 Eclipse 菜单中选择 Window/Preferences/General/Editors/File Associations/选择 *.js,将 Spket JavaScript Editor 设为 Default。如附图 B-7 所示。



附图 B-7　将 SpketIDE 设置为 js 文件的默认编辑器

⑤ JS 开发 IDE 已经安装和配置完成,在工程里面享受智能提示的乐趣吧!

参考文献

[1] Bergsten, H. JSP 设计[M]. 2 版. 汪青青, 朱剑平, 译. 清华大学出版社, 2004.

[2] 克拉恩. Ajax 实战[M]. ajaxcn. org, 译. 人民邮电出版社, 2006.

[3] 陶国荣. jQuery 权威指南[M]. 机械工业出版社, 2013.

[4] 刘志成. JSP 程序设计实例教程[M]. 人民邮电出版社 2009.

策划编辑：罗晓莉
封面设计：runsign 道正设计

基于 BootStrap3 的 JSP 项目实例教程

赠送电子课件

本书配有源代码、课件、教案及相关学习视频
供读者免费试用。

上架建议：计算机语言与程序

ISBN 978-7-5124-1821-9

定价：32.00元